Brief Description

The ZSSC1956 IC is a dual-channel analog-to-digital converter (ADC) with an embedded microcontroller for battery sensing/management in automotive, industrial, and medical systems.

One of the two input channels measures the battery current IBAT via the voltage drop at the external shunt resistor. The second input channel measures the battery voltage VBAT and the temperature. An integrated flash memory is provided for customer-specific software; e.g., dedicated algorithms for calculating the battery state.

During Sleep Mode (e.g., engine is off), the system makes periodic measurements to monitor the discharge of the battery. Measurement cycles are controlled by the software and include various wake-up conditions. The ZSSC1956 is optimized for ultra-low power consumption and draws only 100µA or less in Low-Power Mode.

Features

- High-precision 24-bit sigma-delta ADC (18-bit with no missing codes); sample rate: 1Hz – 16kHz
- On-chip voltage reference (5ppm/K typical)
- Current channel
  - I_{BAT} offset error: ≤ 10mA
  - I_{BAT} resolution: ≤ 1mA
  - Programmable gain: 4 to 512
  - Differential input stage input range: ± 300mV
- Voltage channel
  - Input range: 4 to 28.8V
  - Voltage accuracy: better than ±2mV
- Temperature channel
  - Internal temperature sensor: ± 2°C
  - External temperature sensor (NTC)
- On-chip precision oscillator (1%) and on-chip low-power oscillator
- ARM® Cortex™-M0* microcontroller: 32-bit core, 10MHz to 20MHz
- 96kB Flash/EE Memory with ECC, 8kB SRAM
- LIN2.2 / SAE J2602-2 compliant
- Directly connected to 12V battery supply
- Normal Mode current consumption: 10mA to 20mA
- Low-Power Mode current consumption: ≤ 100µA

Benefits

- Integrated, precision measurement solution for accurate prediction of battery state of health (SOH), state of charge (SOC) or state of function (SOF)
- Flexible wake-up modes allow minimum power consumption without sacrificing performance
- No temperature calibration or external trimming components required
- Optimized code density through small instruction set architecture Thumb®-2 *
- Robust power-on-reset (POR) concept for harsh automotive environments
- Industry's smallest footprint allows minimal module size and cost
- AEC-Q100 qualified solution

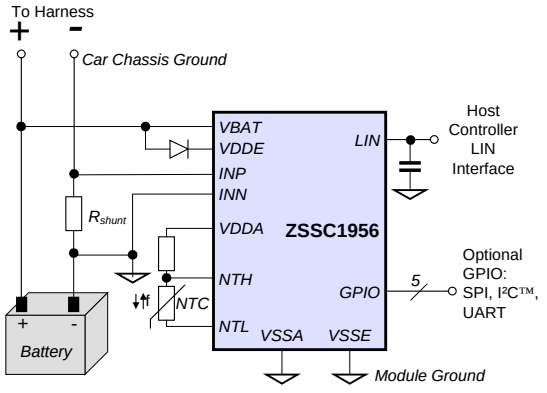
Available Support

- Evaluation Kit
- Application Notes

Physical Characteristics

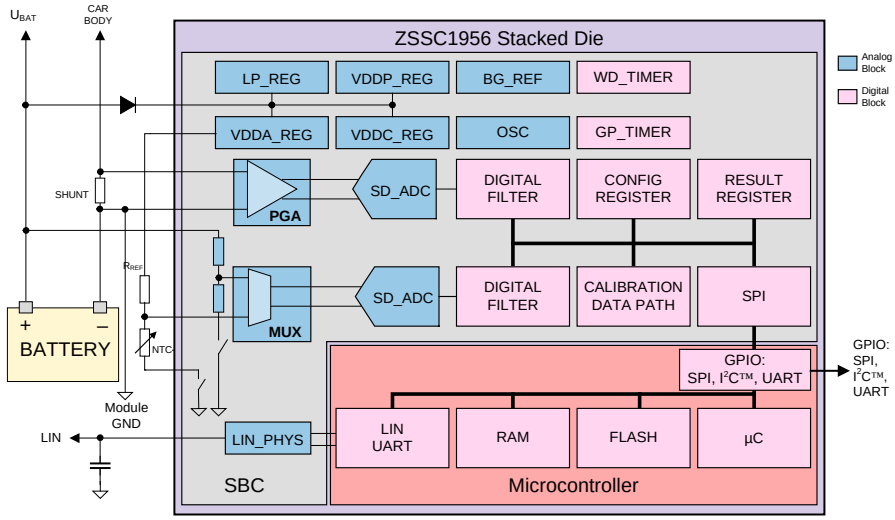
- Wide operation temperature: -40°C to +125°C
- Supply voltage: 4.2 to 18V
- Small footprint package: PQFN32 5x5 mm

Basic ZSSC1956 Application Circuit

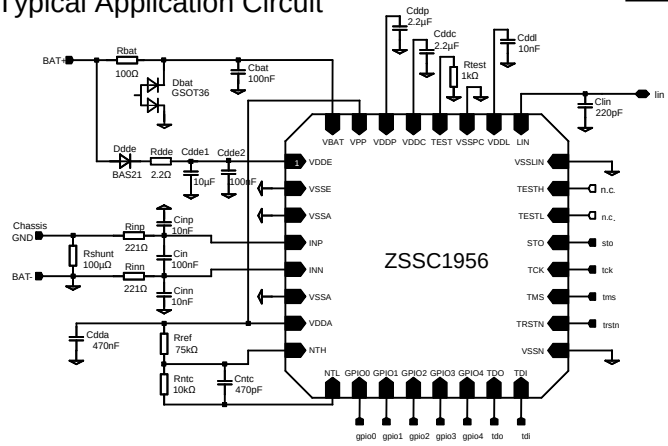


* The ARM®, Cortex™, and Thumb®-2 trademarks are owned by ARM, Ltd. The I²C™ trademark is owned by NXP.

ZSSC1956 Block Diagram



Typical Application Circuit



Applications

- Intelligent battery sensing for automotive applications; e.g., start/stop systems, e-bikes, scooters, and e-carts
- Industrial and medical applications requiring precise battery SOC, SOH and SOF monitoring; e.g., emergency lighting, uninterruptable power supplies, hospital equipment, alarm systems, and more

Ordering Information

Product Sales Code	Description	Package
ZSSC1956BA3R	ZSSC1956 battery sensing IC – temperature range -40°C to +125°C	PQFN32 5x5 mm (reel)
ZSSC1956KIT V1.0	ZSSC1956 Evaluation Kit: modular evaluation and development board for ZSSC1956, IC samples, and USB cable, (software and documentation can be downloaded from the product page at www.IDT.com/ZSSC1956)	

Corporate Headquarters
6024 Silver Creek Valley Road
San Jose, CA 95138
www.IDT.com

Sales
1-800-345-7015 or 408-284-8200
Fax: 408-284-2775
www.IDT.com/go/sales

Tech Support
www.IDT.com/go/support

DISCLAIMER Integrated Device Technology, Inc. (IDT) reserves the right to modify the products and/or specifications described herein at any time, without notice, at IDT's sole discretion. Performance specifications and operating parameters of the described products are determined in an independent state and are not guaranteed to perform the same way when installed in customer products. The information contained herein is provided without representation or warranty of any kind, whether express or implied, including, but not limited to, the suitability of IDT's products for any particular purpose, an implied warranty of merchantability, or non-infringement of the intellectual property rights of others. This document is presented only as a guide and does not convey any license under intellectual property rights of IDT or any third parties.

IDT's products are not intended for use in applications involving extreme environmental conditions or in life support systems or similar devices where the failure or malfunction of an IDT product can be reasonably expected to significantly affect the health or safety of users. Anyone using an IDT product in such a manner does so at their own risk, absent an express, written agreement by IDT.

Integrated Device Technology, IDT and the IDT logo are trademarks or registered trademarks of IDT and its subsidiaries in the United States and other countries. Other trademarks used herein are the property of IDT or their respective third party owners. For datasheet type definitions and a glossary of common terms, visit www.idt.com/go/glossary. All contents of this document are copyright of Integrated Device Technology, Inc. All rights reserved.

Contents

© 2016 Integrated Device Technology, Inc.

© 2016 Integrated Device Technology, Inc.

List of Figures

List of Tables

© 2016 Integrated Device Technology, Inc.

1 IC Characteristics

1.1. Absolute Maximum Ratings

Note: The absolute maximum ratings in section 1.1 are stress ratings only. The device might not function or be operable above the recommended operating conditions given in section 1.2. Stresses exceeding the absolute maximum ratings might also damage the device. In addition, extended exposure to stresses above the recommended operating conditions might affect device reliability. IDT does not recommend designing to the "Absolute Maximum Ratings."

Table 1.1 Absolute Maximum Ratings (referenced to VSSE)

No	Parameter	Symbol	Conditions	Min	Max	Unit
1.1.1.	External power supply	V_{DDE}		V_{SSE}-0.3	40	V
1.1.2.	Current sensing, INP pin	V_{INP}		V_{SSE}-0.3	V_{DDA}+0.3	V
1.1.3.	Current sensing, INN pin	V_{INN}		V_{SSE}-0.3	V_{DDA}+0.3	V
1.1.4.	Voltage sensing, VBAT pin	V_{VBAT}		-18	33	V
1.1.5.	Voltage sensing, VBAT pin	V_{VBAT}	1h over lifetime	-18	40	V
1.1.6.	Temperature sensing, NTH pin	V_{NTH}		V_{SSE}-0.3	V_{DDA}+0.3	V
1.1.7.	Temperature sensing, NTL pin	V_{NTL}		V_{SSE}-0.3	V_{DDA}+0.3	V
1.1.8.	LIN bus interface, LIN pin	V_{LIN}		-16	33	V
1.1.9.	LIN bus interface, LIN pin	V_{LIN}	1h over lifetime	-16	40	V
1.1.10.	GPIO pins	V_{GPIO}		V_{SSE}-0.3	V_{DDP}+0.3	V
1.1.11.	Ambient temperature under bias	T_A			125	°C
1.1.12.	Junction temperature	T_j			135	°C
1.1.13.	Storage temperature	T_S		-50	125	°C

© 2016 Integrated Device Technology, Inc.

1.2. Recommended Operating Conditions

Table 1.2 Operating Conditions

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.2.1	Operating temperature range	T_A	Ambient, RTHP=35K/W	-40		115	°C
1.2.2	Extended temperature range	T_{A_Ext}	Ambient, with reduced accuracies	-40		125	°C
1.2.3	Supply voltage at BAT+ terminal [1] for normal operation	V_{BAT+}		6	13	18	V
1.2.4	Minimum supply voltage at VDDE pin: a) When V_{BAT+} < 6V, i.e. operation with low battery b) When $V_{BAT+} = V_{DDE}$, i.e. without using Ddde and Rdde (see Figure 5.1)	V_{DDE_low}	Normal accuracy for current and temperature measurements Reduced accuracy for voltage measurements	4.8			V
			Reduced accuracy for all measurements	4.2			
1.2.5	Digital input voltage LOW	V_{IL}		0		$0.3*V_{DDP}$	V
1.2.6	Digital input voltage HIGH	V_{IH}		$0.7*V_{DDP}$		V_{DDP}	V

1) See application diagram on page 3.

1.3. Electrical Parameters

Note: See important notes at the end of the following table. See section 3.7 for definitions of power states.

Table 1.3 Electrical Specifications

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Supply							
1.3.1.	Average supply current at VDDE	I_{DDE_avg}	Normal Mode (FP state, both ADC ON)		20		mA
1.3.2.	Average power dissipation	P_{DDE_avg}	Normal Mode, V_{DDE}=13V		260		mW
1.3.3.	Average supply current at VDDE	I_{DDE_slp}	Sleep Mode (ULP state, no measurement)		65		µA
1.3.4.	Average supply current at VDDE	I_{DDE_cmp}	Comparator Mode, ULP state with wake up time interval = 30s, I-ADC only		160		µA
1.3.5.	Average supply current at VDDE	I_{DDE_off}	OFF state (no measurement, transportation mode)		60		µA
1.3.6.	Internal analog power supply voltage	V_{DDA}		2.4	2.5	2.6	V
1.3.7.	Internal digital and RAM power supply voltage	V_{DDL}		1.62	1.8	1.98	V
1.3.8.	Internal power supply voltage for microcontroller unit (MCU) core	V_{DDC}		1.62	1.8	1.98	V
1.3.9.	Internal power supply voltage (periphery)	V_{DDP}		2.97	3.3	3.63	V
Current Channel							
1.3.10.	Input signal range [1)]	$Range_C$	Gain = 4	-300		300	mV
			Gain = 8	-150		150	mV
			Gain = 16	-75		75	mV
			Gain = 32	-38		38	mV
			Gain = 64	-19		19	mV
			Gain = 128	-9.5		9.5	mV
			Gain = 256	-4.7		4.7	mV

© 2016 Integrated Device Technology, Inc.

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
			Gain = 512	-2.3		2.3	mV
1.3.11.	Input leakage current [1]	I_{LEAK_C}	T_A = 25°C	-3		+3	nA
1.3.12.	Input offset current [1]	I_{OFFSET_C}	For input signal < 10mV		0.5	1.5	nA
1.3.13.	Conversion rate [1), 2)]	$Rate_C$	Programmable	1		16000	Hz
1.3.14.	Oversampling ratio [1)] (Sinc^4 decimation filter)	OSR_C	Programmable	32		256	
1.3.15.	No missing codes [1)]	NMC_C		18			Bits
1.3.16.	Integral nonlinearity [1), 3), 4), 5)]	INL	Maximum input range		±10	±60	ppm of FSR
1.3.17.	PGA gain range [1)]	A_{PGA}		4		512	
1.3.18.	Total gain error [1), 8)]	err_{PGA_C}		-1		1	%
1.3.19.	Gain drift [1)]	$err\_drift_{PGA\_C}$			±3		ppm/°C
1.3.20.	Offset error after ZSSC1956 calibration [1), 6)]	V_{OFFSET_C}	Normal Mode chop on, external short (VSSA)	-2		2	µV
			Low-Power Mode, chop on, external short (VSSA)	-0.6		0.6	µV
1.3.21.	Offset error drift [1), 4)]	$V_{OFFSET\_DRIFT\_C}$	Chop on		±20		nV/°C
			Chop off		±80		nV/°C
1.3.22.	Output noise with chop on [1), 10)]	V_{NOISE_C}	Gain = 512, conversion rate = 10Hz		1.1		μV^{RMS}
			Gain = 512, conversion rate = 1kHz		1.1		μV^{RMS}
			Gain = 32, conversion rate = 1kHz		3		μV^{RMS}
			Gain = 4, conversion rate = 1kHz		11		μV^{RMS}
1.3.23.	Current offset [1)]	I_{BAT_offset}	Chop on, gain = 512, R_{shunt} = 100µΩ			10	mA

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.3.24.	Resolution [1]	I_{RES}	Chop on, gain = 512, R_{shunt} = 100µΩ	1			mA
Voltage Channel							
1.3.25.	Input signal range (at V_{BAT} pin) [1]	$Range_V$	Resistive divider (1:24)	0		28.8	V
1.3.26.	Input measurement range [1]	$Range_{meas\_V}$	Resistive divider (1:24)	3.6		28.8	V
1.3.27.	Input valid range for ADC [1]	$Range_{ADC\_V}$	Resistive divider (1:24)	0.15		1.2	V
1.3.28.	Voltage resistive divider ratio [1]	$Ratio_V$			24		
1.3.29.	Resistor divider mismatch drift [1]	$Ratio\_mis_{drift\_v}$			±3		ppm/°C
1.3.30.	Conversion rate [1], [2]	$Rate_V$	Programmable	1		16000	Hz
1.3.31.	Oversampling ratio ($Sinc^4$ decimation filter) [1]	OSR_V	Programmable	32		256	
1.3.32.	No missing codes [1]	NMC_V		18			Bits
1.3.33.	Integral nonlinearity [1], [3], [7]	INL_V	Maximum input range		±10	±60	ppm of FSR
1.3.34.	Total gain error [1] (includes resistor divider mismatch)	err_{PGA_V}		-0.25		0.25	%
1.3.35.	Gain drift [1]	$err\_drift_{PGA\_V}$			±3		ppm/°C
1.3.36.	Offset error after calibration: Normal Mode [1], [9]	V_{OFFSET_V}	Chop on external short (1.25V)		200		µV
			Chop off external short (1.25V)		1		mV
1.3.37.	Offset error drift [1]	$V_{OFFSET\_DRIFT\_V}$	Chop on		±10		µV/°C
			Chop off		±20		µV/°C

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.3.38.	Output noise [1), 10)]	V_{NOISE_V}	Chop on gain = 1, conversion rate =10Hz		30	50	μV^{RMS}
			Chop on gain = 1, conversion rate = 1kHz		1		mV^{RMS}
Temperature Channel (External NTC/Reference Resistor)							
1.3.39.	Voltage drop over NTC resistor [1)]	V_{NTC}		0		1.2	V
1.3.40.	Voltage drop over reference resistor [1)]	V_{REF_Res}		0		1.2	V
1.3.41.	Conversion rate [1)]	$Rate_T$	Programmable	1		16000	Hz
1.3.42.	Oversampling ratio ($Sinc^4$ decimation filter) [1)]	OSR_T	Programmable	32		256	
1.3.43.	Integral nonlinearity [1), 3)]	INL_T	Maximum input range		±10	±60	ppm of FSR
1.3.44.	No missing codes [1)]	NMC_T		16			Bit
1.3.45.	Offset error after ZSSC1956 calibration [1), 9)]	V_{OFFSET_T}	Normal Mode, chop on, external short (1.25V)	-100		100	µV
			Normal Mode, chop off, external short (1.25V)	-2		2	mV
1.3.46.	Offset error drift [1)]	$V_{OFFSET\_drift\_T}$	Chop on		±10		$\mu V/^oC$
			Chop off		±20		$\mu V/^oC$
1.3.47.	Output noise [1), 10)]	V_{NOISE_T}	Chop on, gain = 1, conversion rate =500Hz		30	50	μV^{RMS}
1.3.48.	Resistor to ground at NTL pin [1)]	GND_{RES}			120		kΩ
1.3.49.	Linearity error of internal temperature sensor			-2		2	°C

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.3.50.	Resolution of internal temperature sensor		Factory calibrated (self-heating is not included; it must be calculated and taken into account)		1/32		°C/LSB
Power-on Reset (POR)							
1.3.51.	Power-on reset	V_{PORB}	At V_{DDE}	2.75	3.0	3.6	V
1.3.52.	Power-on-reset hysteresis	$Hyst_{PORB}$	At V_{DDE}		300		mV
1.3.53.	Low-voltage flag	low_voltage	At V_{DDE}	1.8	2.0	2.3	V
1.3.54.	V_{DDP} high [1)]	vddp_high	At V_{DDE}	3.9	4.05	4.2	V
1.3.55.	V_{DDP} high hysteresis [1)]	$Hyst_{VDDP\_high}$	At V_{DDE}		400		mV
Low-Power Voltage Reference							
1.3.56.	Reference bandgap voltage: low-power	V_{BGL}		1.16		1.32	V
1.3.57.	Accuracy (including temperature drift)			-3		3	%
1.3.58.	Temperature coefficient [1)]	TC_{VBGL}			50		ppm/K
Low-Power (LP) Oscillator							
1.3.59.	Frequency	f_{LPO}			125		kHz
1.3.60.	Accuracy (including temperature drift) [1)]			-3		3	%
High-Precision Voltage Reference							
1.3.61.	Reference bandgap voltage: high-precision	V_{BGH}	Uncalibrated	1.16		1.32	V
1.3.62.	Temperature coefficient [1)]	TC_{VBGH}	Calibrated	-20	±5	+20	ppm/K
High-Precision (HP) Oscillator							
1.3.63.	Frequency	f_{HPO}			20		MHz
1.3.64.	Accuracy (including temperature drift) [1)]			-1		1	%
LIN Interface							
1.3.65.	Current limitation for driver dominant state [1), 11)]	I_{BUS_LIM}	LIN spec 2.1 Param 12	40		200	mA
1.3.66.	Input leakage current, dominant state, driver off [1), 11)]	$I_{BUS\_PAS\_dom}$	LIN spec 2.1 Param 13	-1			mA

© 2016 Integrated Device Technology, Inc.

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.3.67.	Input leakage current, recessive state, driver off [1), 11)]	$I_{BUS\_PAS\_rec}$	LIN spec 2.1 Param 14			20	µA
1.3.68.	Control unit disconnected from ground [1), 11)]	$I_{BUS\_NO\_GND}$	LIN spec 2.1 Param 15	-1		1	mA
1.3.69.	V_{BAT} disconnected [1), 11)]	$I_{BUS\_NO\_BAT}$	LIN spec 2.1 Param 16			100	µA
1.3.70.	Receiver dominant state, V_{DDE} > 7V [1), 11)]	V_{BUSdom}	LIN spec 2.1 Param 17			0.4	V_{DDE}
1.3.71.	Receiver recessive state, V_{DDE} > 7V [1), 11)]	V_{BUSrec}	LIN spec 2.1 Param 18	0.6			V_{DDE}
1.3.72.	Center of receiver threshold [1), 11)]	V_{BUS_CNT}	LIN spec 2.1 Param 19	0.475	0.5	0.525	V_{DDE}
1.3.73.	Receiver hysteresis voltage [1), 11)]	V_{HYS}	LIN spec 2.1 Param 20			0.175	V_{DDE}
1.3.74.	Voltage drop at serial diodes [1), 11)]	$V_{SerDiode}$	LIN spec 2.1 Param 21	0.4	0.7	1	V
1.3.75.	Battery shift [1), 11)]	V_{Shift_BAT}	LIN spec 2.1 Param 22			0.115	V_{BAT}
1.3.76.	Ground shift [1), 11)]	V_{BUS_GND}	LIN spec 2.1 Param 23			0.115	V_{BAT}
1.3.77.	Difference between battery shift and ground shift [1), 11)]	$V_{SHIFT\_Difference}$	LIN spec 2.1 Param 24	0		8	%
1.3.78.	LIN pull-up resistor [1), 11)]	R_{SLAVE}	LIN spec 2.1 Param 26	20	30	47	kΩ
1.3.79.	Duty cycle 1 [1), 11)]	D1	LIN spec 2.1 Param 27	0.396			
1.3.80.	Duty cycle 2 [1), 11)]	D2	LIN spec 2.1 Param 28			0.581	
1.3.81.	Duty cycle 3 [1), 11)]	D3	LIN spec 2.1 Param 29	0.417			
1.3.82.	Duty cycle 4 [1), 11)]	D4	LIN spec 2.1 Param 30			0.590	

© 2016 Integrated Device Technology, Inc.

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
1.3.83.	Receiver propagation delay [1), 11)]	t_{RX_pdr}	LIN spec 2.1 Param 31			6	µs
1.3.84.	Symmetry receiver propagation delay, rising/falling edge [1), 11)]	t_{RX_sym}	LIN spec 2.1 Param 32	-2		2	µs
1.3.85.	Capacitance of slave node [1), 11)]	C_{SLAVE}	LIN spec 2.1 Param 37			250	pF
1.3.86.	LIN pin capacitance [1), 11)]	C_{LIN}		-	-	30	pF
Microcontroller Platform							
1.3.87.	MCU start-up time after Sleep Mode wake up / POR [1)]				80		µs
1.3.88.	MCU start-up time after Standby Mode wake up [1)]				1		µs
eFlash Memory							
1.3.89.	Memory size [1)]				96		kB
1.3.90.	Page size [1)]				512		B
1.3.91.	Access time [1)]					35	ns
1.3.92.	Read current [1)]					15	mA
1.3.93.	Page erase time [1)]			16		24	ms
1.3.94.	Mass erase time [1)]			16		24	ms
1.3.95.	Endurance [1)]			10k			cycles
1.3.96.	Data retention time [1)]			10			years
SRAM							
1.3.97.	Memory size [1)]				8		kB
Timer 0 (Sleep Timer)							
1.3.98.	Time interval [1)]	SLPTI1	Programmable	0.1		6553.5	s
1.3.99.	Resolution [1)]	$SLPTI1_{res}$			100		ms
1.3.100.	Time interval with post-scaler [1)]	SLPTI2	Programmable			466	h

No.	Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Timer 1 (Watchdog Timer)							
1.3.101.	Time interval [1)]	WDTI	Programmable	8µ		6553.5	s
1.3.102.	Resolution [1)]	$WDTI_{RES}$	Programmable	0.008		100	ms

1) Not tested in production test; given by design and/or characterization.
2) Conversion rate depends on chopping and OSR settings.
3) Full-scale range (FSR) = 1.2V.
4) For gain setting up to 64.
5) Minimum input voltage I-ADC = -50mV for gain setting 4, 8, 16.
6) Gain setting 16 to 512.
7) Voltage range 7V to 19V.
8) Valid for gain=4, factory calibrated at gain=4, calibration data stored in OTP.
9) Error included in total gain error.
10) Noise can be further reduced by applying averaging.
11) Valid with no additional series resistor.

© 2016 Integrated Device Technology, Inc.

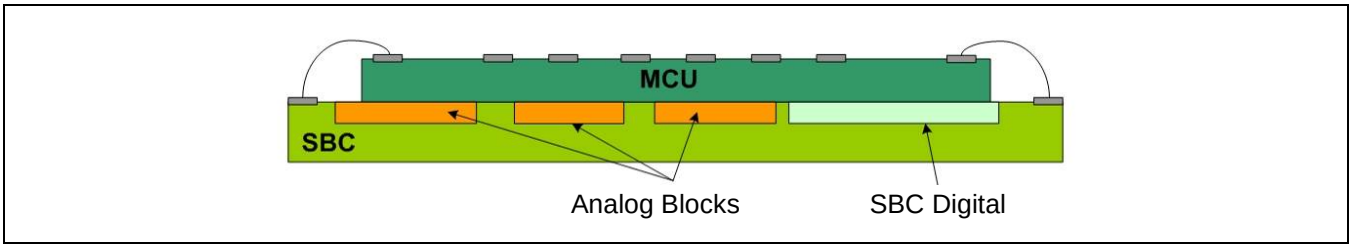
2 Circuit Description

2.1. Overview

The ZSSC1956 intelligent battery sensor IC (IBS) consists of two silicon dies in one package. The dies are assembled as stacked dies in a PQFN32 5x5 mm package as shown in Figure 2.1. See Figure 2.2 for the ZSSC1956 block diagram. The System Basis Chip (SBC) contains the high voltage circuits, analog input stage including peripheral blocks, ΣΔ-ADCs (SD_ADC), digital filtering, and LIN transceiver. The microcontroller chip (MCU) contains the microcontroller core, memories, and some peripheral blocks. Communication between the MCU and the SBC is handled by an SPI interface. Internal nodes connecting the MCU and the SBC (i.e., TXD, RXD, IRQN, CSN, SCLK, MOSI, MISO, MCU_CLK, MCU_RSTN, and RAM_PROTN) are controlled by firmware. Users can access the internal nodes via the LIN interface and/or external JTAG pins (i.e., TDO, TDI, TRSTN, TMS, and TCK).

The functions of the SBC are controlled by register settings. The circuit starts after power-on with default register and calibration settings that can be overwritten by the user software.

Figure 2.1 IBS Stacked Die Assembly



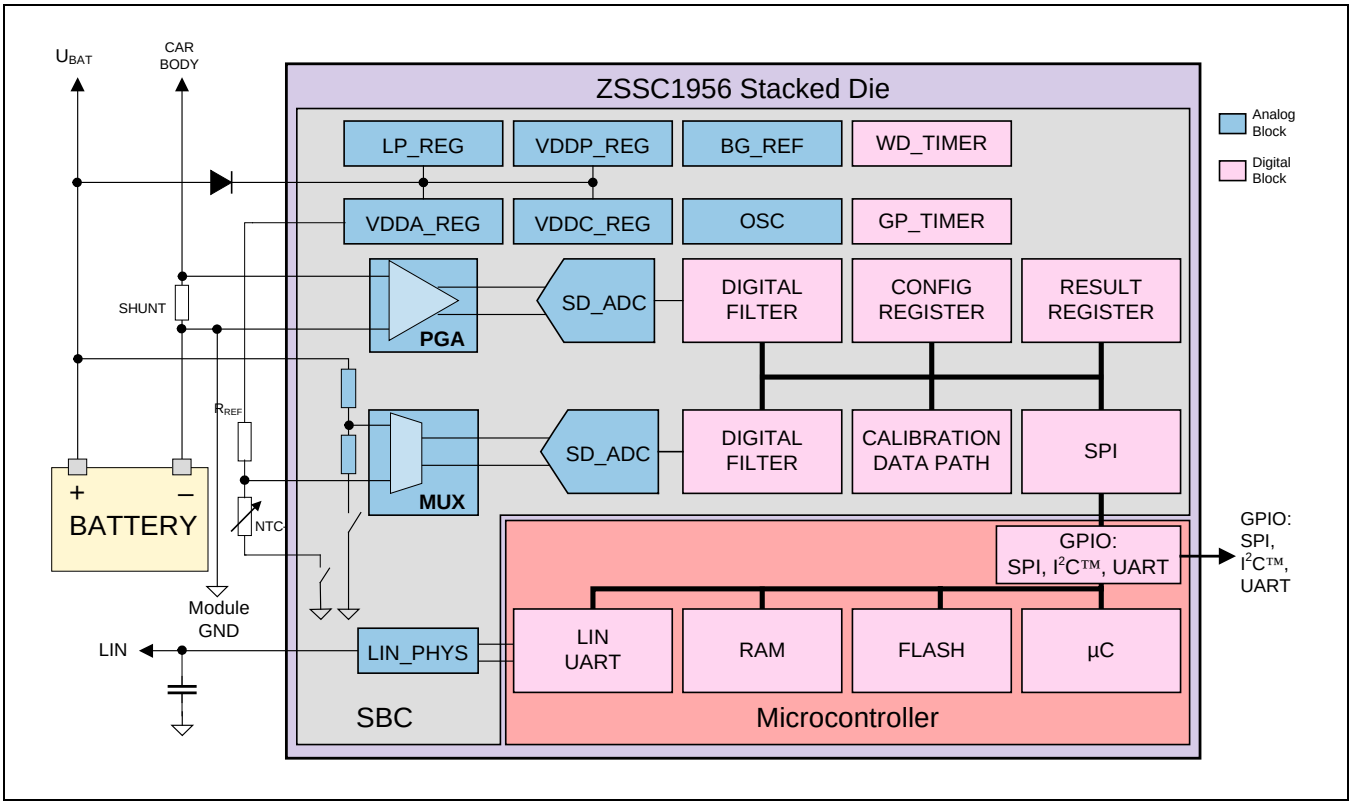
One input channel measures I_{BAT} via the voltage drop at the external shunt resistor. The second channel measures V_{BAT} and the temperature. By simultaneous measurement of V_{BAT} and I_{BAT}, it is possible to determine dynamically R_{BAT}, which is correlated with the state of health (SOH) of the battery. By integrating I_{BAT}, it is possible to determine the state of charge (SOC) of the battery. These are the fundamental parameters for an intelligent battery sensor. The software for determining these parameters is not part of the ZSSC1956. A flash memory is provided for the customer-specific software.

The SBC and MCU have different power states designed to minimize power consumption (see section 2.4).

During the Standby and the Sleep Modes (e.g., engine is off), the system periodically measures the values to monitor the discharge of the battery. Measurement cycles are controlled by the software and are dependent on the detected events. The ZSSC1956 is designed for low-power consumption during Sleep Mode in the range of less than 100µA.

© 2016 Integrated Device Technology, Inc.

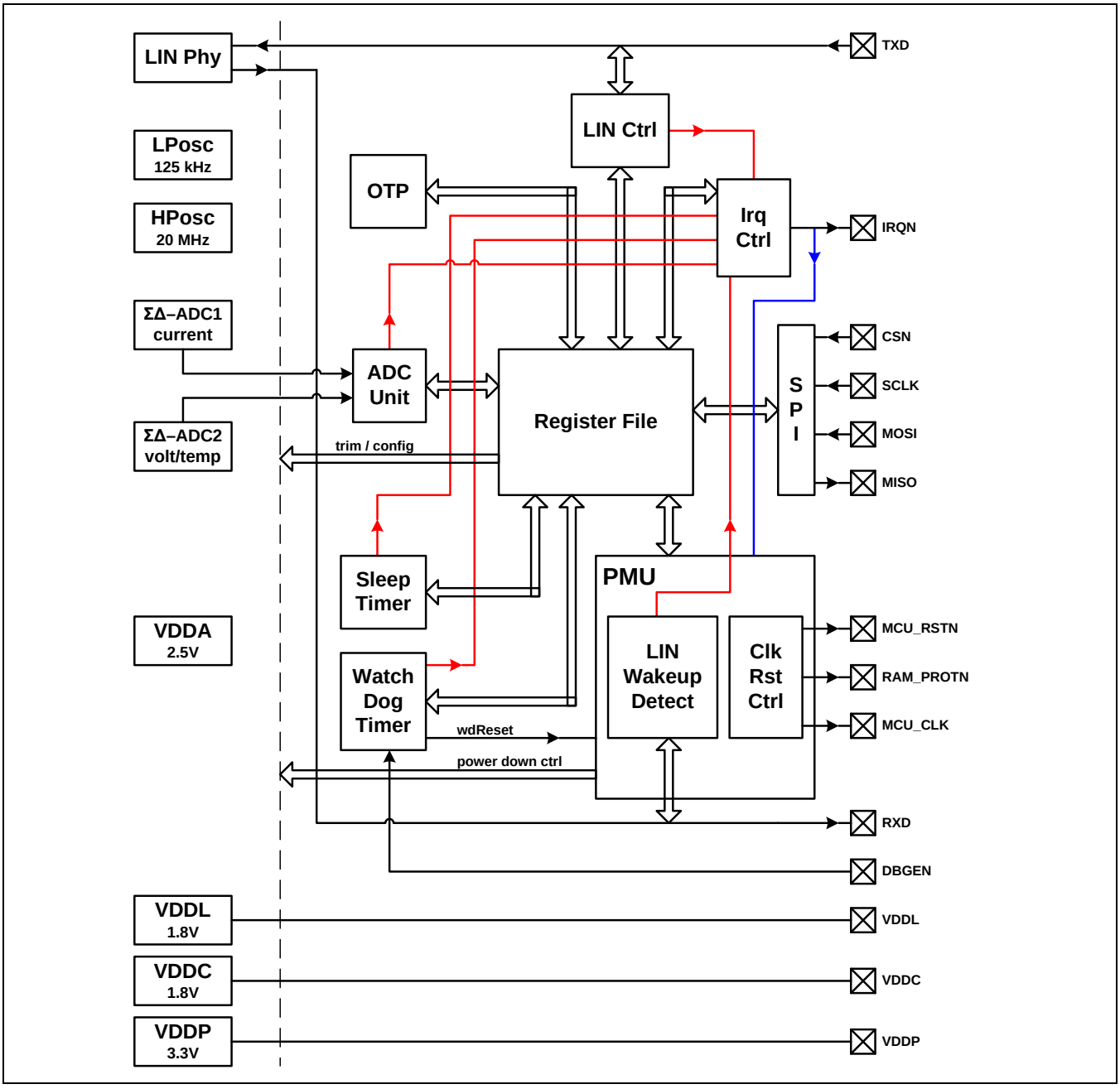
Figure 2.2 Functional Block Diagram



2.2. Digital Block Diagram SBC

In Figure 2.3, the red lines indicate set-interrupt signal paths and the blue lines indicate wake-up signal paths. TXD, IRQN, CSN, SCLK, MOSI, MISO, MCU_RSTN, RAM_PROTN, MCU_CLK, RXD, and DBGEN are internal nodes.

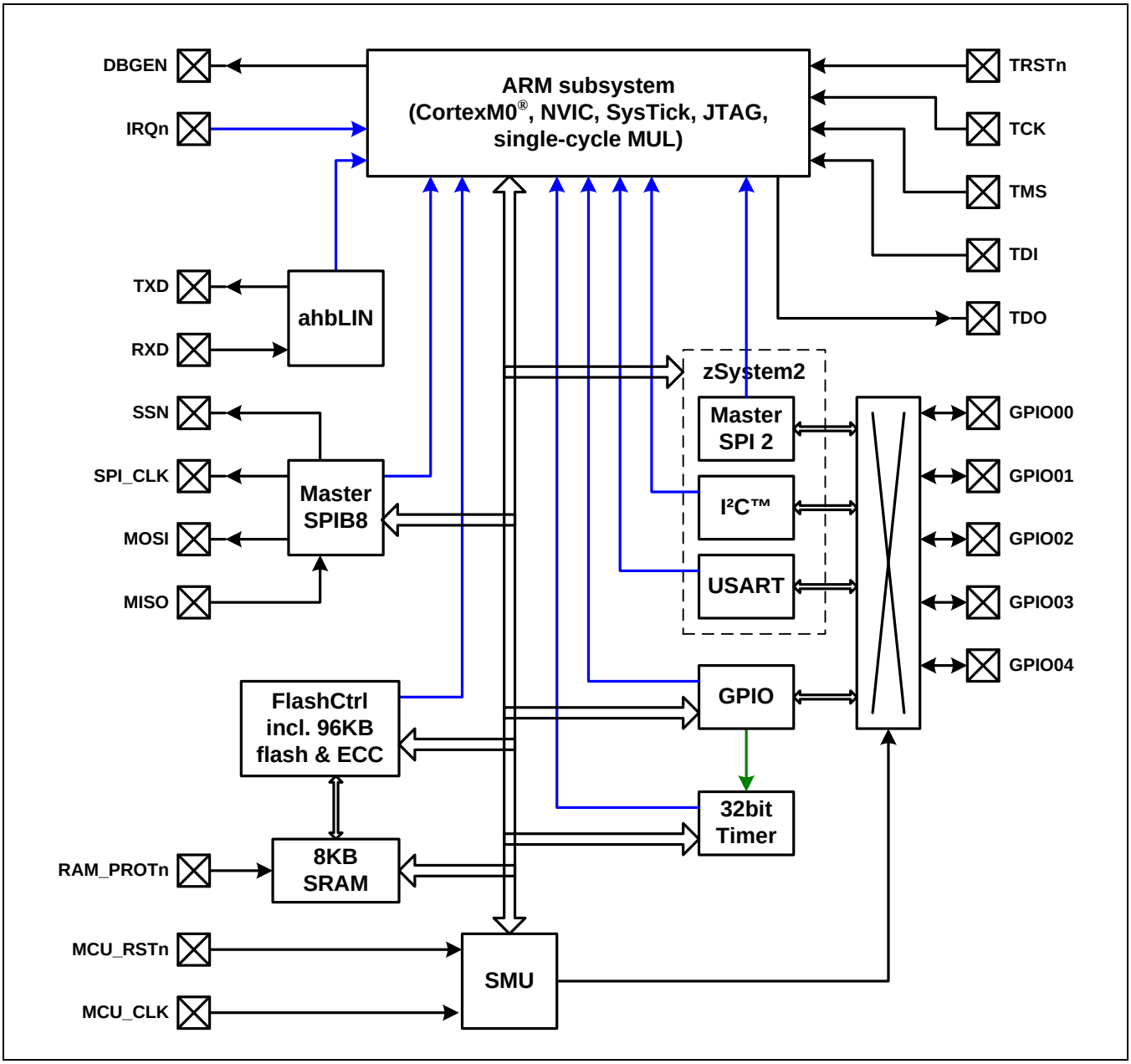
Figure 2.3 Block Diagram of the Digital Section of the SBC



2.3. Block Diagram MCU

In Figure 2.5, the blue lines indicate interrupt signals and the green line indicates trigger signals. The nodes on the left are internal, and the pads on the right are external.

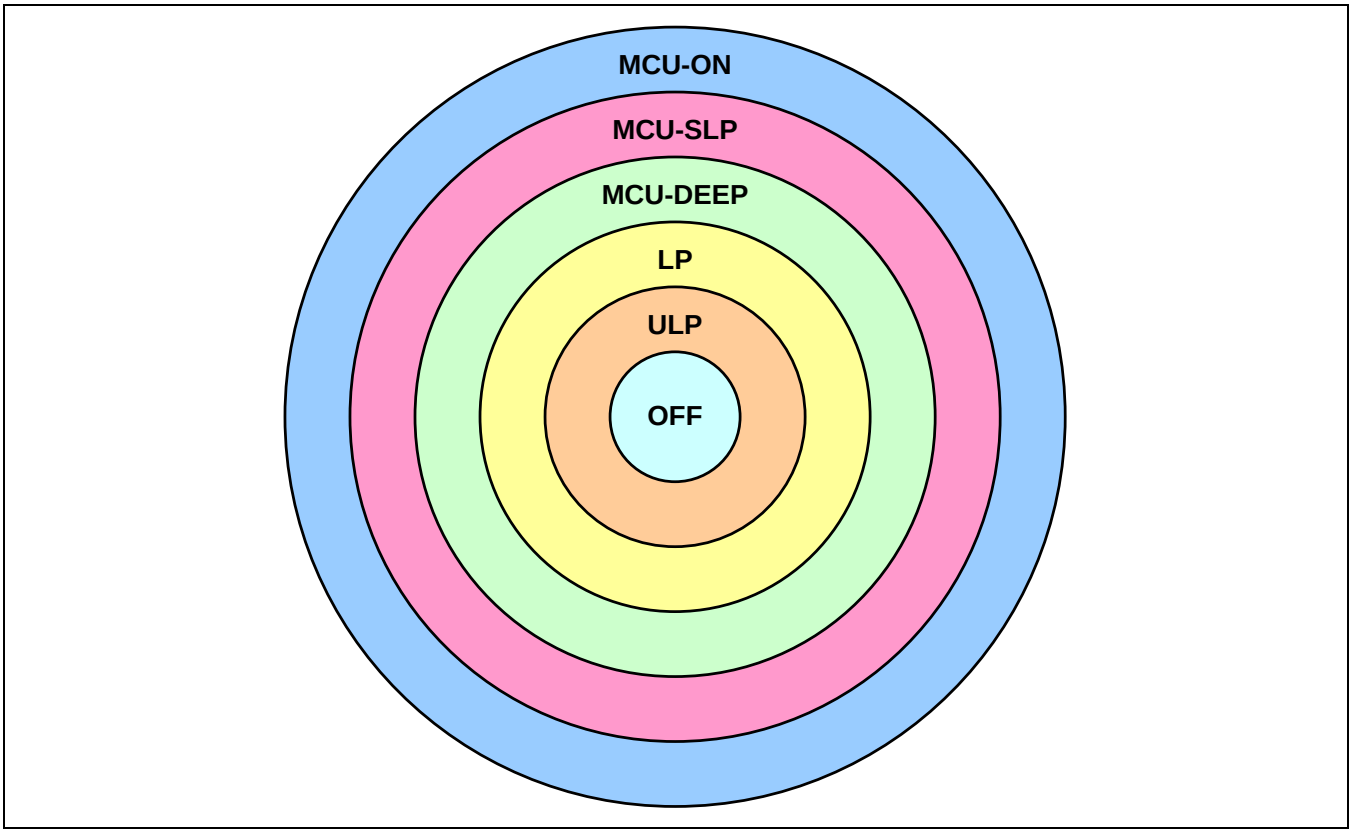
Figure 2.4 Block Diagram of the MCU



2.4. System Power States

Six different power states are implemented in the ZSSC1956, which are combinations of the different power states of the SBC (see section 3.7) and of the MCU (see “Power Modes” in section 4.3). As discussed in the following sections, other combinations are not allowed.

Figure 2.5 System Power States



2.4.1. MCU-ON Power State

The MCU-ON power state is entered after power-on reset or after wake-up. In this power state, the MCU is in its Normal Mode and the SBC is in its Full-Power (FP) state (see section 3.7). The SBC powers the MCU and generates the 20MHz clock for the MCU, which is distributed inside the MCU to the ARM® core as well as to the peripherals. The base clock for the ADCs in the SBC is 4MHz, which is generated from the 20MHz from the high-precision oscillator.

In this state, the ARM® core is running and executing the software from flash or RAM. The software can trigger the system management unit (SMU) (see section 4.3) in the MCU as well as the power management unit (PMU) (see section 3.7) in the SBC to enter any other power state.

Of the six power states, the MCU-ON state consumes the most power.

2.4.2. MCU-SLP Power State

In MCU-SLP power state, the SBC remains in its FP state, which means that it still powers the MCU and generates the 20MHz clock for the MCU. Inside the MCU, all peripherals are clocked, but the clock for the ARM® core is stopped. This power state is intended for scenarios where the ARM® core does not need to execute code but waits for MCU internal peripherals to finish their programmed tasks; e.g., sending a byte via the LIN interface. The system can wake up from this power state by an enabled interrupt as well as by an MCU reset generated by the SBC.

Note: For any SBC interrupt source that will wake up the system, the corresponding interrupt source must be enabled in the SBC, and the ARM® interrupt line 1 (SBC interrupt) must be enabled in the MCU's interrupt controller (NVIC) (see section 4.1).

When the system will enter the MCU-SLP power state from the MCU-ON power state, the software must first enable the required interrupt sources in the SBC and the MCU and it must set the SLEEPDEEP bit in the ARM® internal system control register (SCR) to 0 (see the *IDT ARM® Cortex™ M0 User Guide*). Then the software must execute the wait-for-interrupt (WFI) instruction to enter the MCU-SLP power state. When any of the enabled interrupts becomes active, the system returns to the MCU-ON power state and continues the software execution.

Note: Do not send a power-down command to the SBC in advance.

2.4.3. MCU-DEEP Power State

In MCU-DEEP power state, the SBC remains in its FP state, which means that it still powers the MCU and generates the 20MHz clock for the MCU. Inside the MCU, the incoming clock is gated, which means that no logic in the MCU is clocked. This power state is intended for conditions where the SBC will perform high-precision measurements using the 4MHz clock as the base clock for the ADCs, but the ARM® core will not run its software and no MCU peripheral is needed. The system can wake up from this power state by an enabled interrupt of the SBC as well as by an MCU reset generated by the SBC.

Note: For any SBC interrupt source that will wake up the system, the corresponding interrupt source must be enabled in the SBC, and the ARM® interrupt line 1 (SBC interrupt) must be enabled in the NVIC.

When the system will enter the MCU-DEEP power state from the MCU-ON power state, the software must first enable the required interrupt sources in the SBC and the MCU and it must set the SLEEPDEEP bit in the ARM® internal SCR register to 1. Then the software must execute the WFI instruction to enter the MCU-DEEP power state. When any of the enabled interrupts becomes active, the system returns to the MCU-ON power state and continues the software execution.

Note: Do not send a power-down command to the SBC in advance.

2.4.4. LP Power State

The LP power state is the first state where the SBC leaves its FP state. The MCU first enters its DEEPSLEEP state, but then the SBC stops the MCU clock on the MCU side and disables the high-precision oscillator. For its ADC operations, the SBC uses the 125kHz clock from the low-power oscillator as the base clock. This power state is intended for conditions where the SBC will only perform low-power measurements without any operation by the MCU. The system can wake up from this power state by an enabled interrupt of the SBC as well as by an MCU reset generated by the SBC.

© 2016 Integrated Device Technology, Inc.

Note: For any SBC interrupt source that will wake up the system, the corresponding interrupt source must be enabled in the SBC and the ARM® interrupt line 1 (SBC interrupt) must be enabled in the NVIC. The latter is necessary as the SBC rejects the power-down command when an enabled interrupt source in the SBC is already active as well as for a final wake up.

When the system will enter the LP power state from the MCU-ON power state, the software must first enable the required interrupt sources in the SBC and MCU and it must set the SLEEPDEEP bit in the ARM® internal register SCR to 1. After successful transmission of the corresponding power-down command to the SBC without releasing the CSN line afterward, the software must then execute the WFI instruction to enter the LP power state. The CSN line is released by hardware on entering the MCU internal DEEPSLEEP state. The rising edge on the CSN line triggers the SBC to enter its LP state. When any of the enabled interrupts becomes active, the system returns to the MCU-ON power state and continues the software execution.

Note: Do not release the CSN line by software at the end of sending a power-down command to avoid the MCU clock being stopped by SBC at an intermediate state.

2.4.5. ULP Power State

The ULP power state is similar to the LP state except that the SBC also disables the power for the MCU. The MCU first enters its DEEPSLEEP state but then the SBC stops the MCU clock on the MCU side and disables the high-precision oscillator and the voltage regulators for the MCU. For its ADC operations, the SBC uses the 125 kHz clock from the low-power oscillator as its base clock. This power state is intended for conditions where the SBC will only perform low-power measurements without any operation on the MCU. The system can wake up from this power state by any enabled interrupt of the SBC. The MCU is reset upon wake up by the SBC to guarantee correct start up. This means that the software starts again from address 00_{HEX} after wake up, not at the position where it was stopped.

Note: For any SBC interrupt source that will wake up the system, the corresponding interrupt source must be enabled in the SBC and the ARM® interrupt line 1 (SBC interrupt) must be enabled in the NVIC. The latter is necessary as the SBC rejects the power-down command when an enabled interrupt source in the SBC is already active.

When the system will enter the ULP power state from the MCU-ON power state, the software must first enable the required interrupt sources in the SBC and MCU and it must set the SLEEPDEEP bit in the ARM® internal register SCR to 1. After successful transmission of the corresponding power-down command to the SBC without releasing the CSN line afterward, the software must then execute the WFI instruction to enter the ULP power state. The CSN line is released by hardware on entering the MCU internal DEEPSLEEP state. The rising edge on the CSN line triggers the SBC to enter its ULP state. When any of the enabled interrupts becomes active, the system returns to MCU-ON power state and restarts the software execution.

Note: Do not release the CSN line by software at the end of sending a power-down command to avoid the MCU clock being stopped by SBC at an intermediate state.

© 2016 Integrated Device Technology, Inc.

2.4.6. OFF Power State

The OFF state is the state with the lowest power consumption where no measurements can be performed as all oscillators are stopped. The MCU first enters its DEEPSLEEP state but then the SBC stops the MCU clock on the MCU side and disables both oscillators and the voltage regulators for the MCU. This power state is intended for conditions where two measurements will be performed and the system will consume as little power as possible. The system can wake up from this power state only by receiving a wakeup frame over LIN. The MCU is reset upon wake up by the SBC to guarantee correct start up. This means that the software starts again from address 00_{HEX} after wake up, not at the position where it was stopped.

Note: For any SBC interrupt source that will wake up the system, the corresponding interrupt source (LIN wakeup interrupt) must be enabled in the SBC and the ARM® interrupt line 1 (SBC interrupt) must be enabled in the NVIC. The latter is necessary as the SBC rejects the power-down command when an enabled interrupt source in the SBC is already active.

When the system will enter the OFF power state from the MCU-ON power state, the software must first enable the required interrupt sources in the SBC and MCU and it must set the SLEEPDEEP bit in the ARM® internal register SCR to 1. After successful transmission of the corresponding power-down command to the SBC without releasing the CSN line afterward, the software must then execute the WFI instruction to enter the OFF power state. The CSN line is released by hardware on entering the MCU internal DEEPSLEEP state. The rising edge on the CSN line triggers the SBC to enter its ULP state. When any of the enabled interrupts becomes active, the system returns to the MCU-ON power state and restarts the software execution.

Note: Do not release the CSN line by software at the end of sending a power-down command to avoid the MCU clock being stopped by the SBC at an intermediate state.

© 2016 Integrated Device Technology, Inc.

3 Functional Block Descriptions SBC

3.1. SPI Communication between the MCU and SBC

The SBC is fully controllable by the MCU via an integrated four-wire slave SPI interface. It only operates in a single mode when both the clock polarity and the clock phase are 1 (the clock is high when inactive, data is sent on the falling SPI clock edge, and data is sampled on the rising SPI clock edge). The accessible registers of the SBC can be read and/or written via the SPI, and the one-time programmable (OTP) memory in the SBC can be read via the SPI. Internal status information of the SBC is also returned during the address and length bytes of the implemented SPI protocol. Read and write burst accesses of up to 128 bytes are supported.

The SPI chip-select line CSN must be low during any transfer until the complete transfer has finished. This is needed as the CSN input is not only used as an enable signal but also as an asynchronous reset for part of the SPI front-end. The reason for this is to be able to set the SPI back to a defined state via the MCU as well as to extract status information without needing to access any register. The CSN input can be kept low between two transfers. The CSN input must only be driven high for execution of the “go-to-power-down” command after the required register settings have been completed.

Note: A high level on the CSN input resets the internal SPI state machine.

3.1.1. SPI Protocol

The SPI slave module only operates with a clock polarity setting of 1 (SCLK is high when no transfer is active; see CPOL in Table 4.49) and with a clock phase setting of 1 (data is sent on the falling edge; data is sampled on the rising edge; see CPHA in Table 4.49). For any access, the CSN input must be low. At the end of any read access, the CSN input can be kept low. For write accesses that change the power mode, the CSN input must be driven high at the end of the write access; it can be kept low for write accesses to other registers. During an SPI access, the CSN input must be kept low.

Important: Driving the CSN input high during a read transfer can cause a loss of data.

In each SPI transfer, 1 to 128 bytes can be read or written in one burst access. All bytes are sent and received with the MSB first. As shown in Figure 3.1, each SPI transfer starts with two bytes sent by the master while the slave sends back status information in parallel. The first of the two bytes sent by the master is the address byte containing the first address to be accessed. When multiple bytes are read or written, the received SPI address is internally incremented for each data byte. The second byte starts with the access type of the transfer (1 = write; 0 = read) followed by the 7-bit length field indicating the number of data bytes that will be read or written. A special case is the length value of 0, which is interpreted by the slave SPI as 128 bytes.

The status information sent back by the slave during the address and length bytes starts with a fixed value of A_{HEX}. This can be used to detect whether the connection is still present. The next bits sent are the slave status word (SSW), which is 12 bits of actual status information.

The 12 SSW bits have the following definitions:

SSW[11]:	Value of the low-voltage flag
SSW[10:8]:	Reset status
SSW[7]:	Watchdog active flag
SSW[6]:	Low-power oscillator trimming circuit active
SSW[5]:	Voltage/temperature ADC active
SSW[4]:	Current ADC active

SSW[3]: LIN short protection active

SSW[2]: LIN TXD timeout protection active

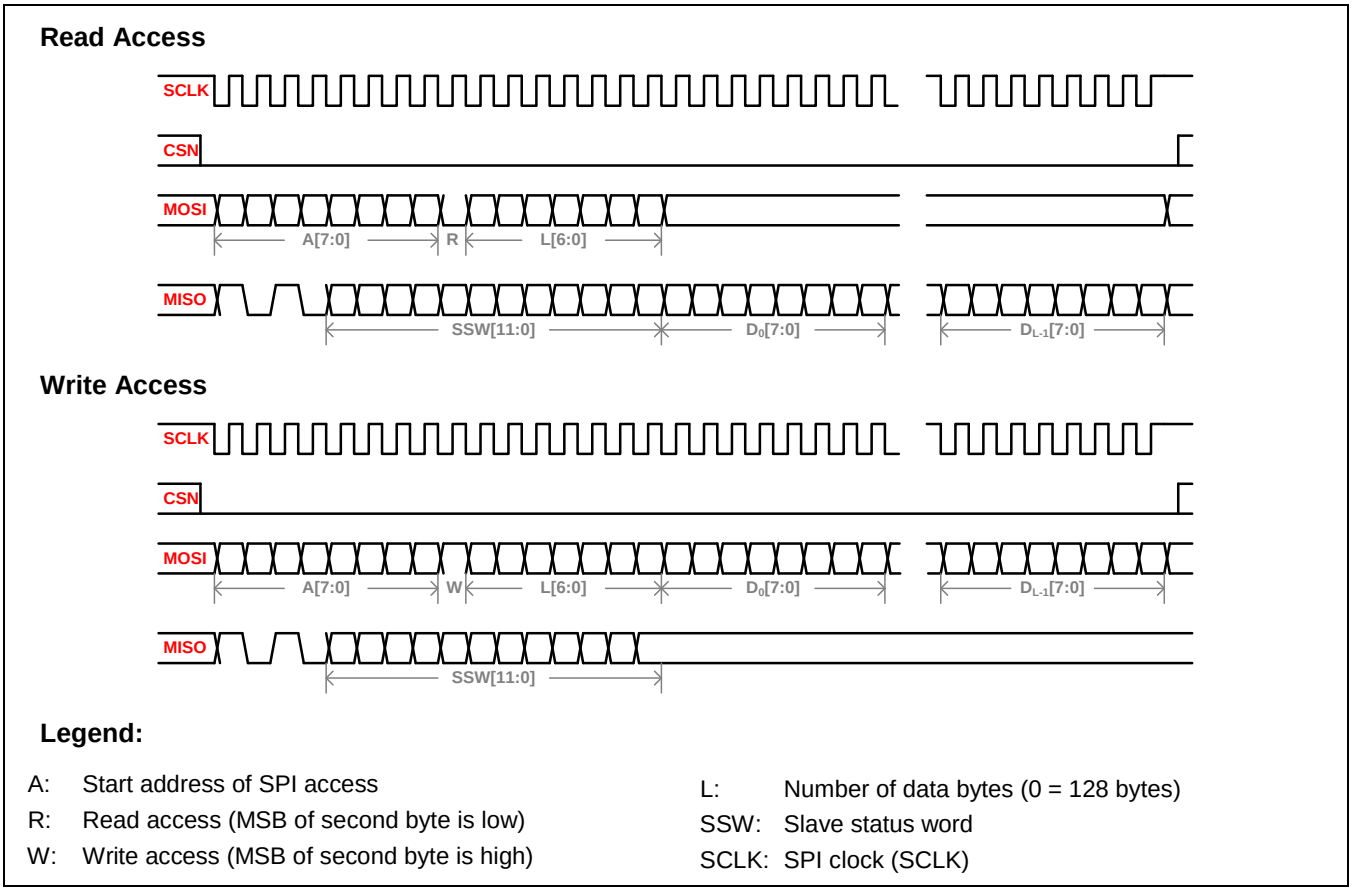
SSW[1]: Readable sleep timer value valid

SSW[0]: OTP download procedure active

Note: After the MCU has been reset, the user's software can read the low-voltage flag and the reset status by a single-byte transfer (**important:** send address byte only) to shorten the initialization phase (e.g., when a reset was caused by a wake-up event) without needing to read or write further bytes including the length byte.

After the address byte and length byte are sent by the master, either the master (write transfer) or the slave (read transfer) is transmitting data. The slave ignores all incoming bits while it is sending the requested number of data bytes (read), and the data bytes returned during a write transfer have no meaning. Figure 3.1 shows a read and a write burst access to the SBC.

Figure 3.1 Read and Write Burst Access to the SBC



3.2. SBC Register Map

Table 3.1 defines the registers in the SBC. In the “Access” column, the following abbreviations indicate the read/write status of the registers: RC = read-clear; RO = read-only; RW = readable and writable; WO = write-only; W1C = write-one-to-clear. For more details, see the subsequent sections for the individual registers.

Important: There is a distinction between “unused” and “reserved” addresses. No problem occurs when writing to unused addresses, but writing 00_{HEX} to unused addresses for future expansions is recommended. Reserved addresses must always be written with the given default value.

Table 3.1 SBC Register Map

Name	Address	Order	Default	Access	Short Description
irqStat	00_{HEX}	LSB	00_{HEX}	RC	Interrupt status register
	01_{HEX}	MSB	00_{HEX}	RC	
adcCdat	02_{HEX}	LSB	00_{HEX}	RO	ADC result register of a single current measurement
	03_{HEX}	---	00_{HEX}	RO	
	04_{HEX}	MSB	00_{HEX}	RO	
adcVdat	05_{HEX}	LSB	00_{HEX}	RO	ADC result register of a single voltage measurement
	06_{HEX}	---	00_{HEX}	RO	
	07_{HEX}	MSB	00_{HEX}	RO	
adcRdat	08_{HEX}	LSB	00_{HEX}	RO	ADC result register of a single temperature measurement by reading a voltage across the reference resistor (external temperature measurement only)
	09_{HEX}	MSB	00_{HEX}	RO	
adcTdat	$0A_{HEX}$	LSB	00_{HEX}	RO	ADC result register of a single temperature measurement by reading a voltage across the NTC resistor (external temperature measurement) or a differential voltage (VPTAT – VBGH; internal temperature measurement)
	$0B_{HEX}$	MSB	00_{HEX}	RO	
adcCaccu	$0C_{HEX}$	LSB	00_{HEX}	RO	Accumulator register for current measurements
	$0D_{HEX}$	---	00_{HEX}	RO	
	$0E_{HEX}$	---	00_{HEX}	RO	
	$0F_{HEX}$	MSB	00_{HEX}	RO	
adcVaccu	10_{HEX}	LSB	00_{HEX}	RO	Accumulator register for voltage measurements
	11_{HEX}	---	00_{HEX}	RO	
	12_{HEX}	MSB	00_{HEX}	RO	
adcCmax	13_{HEX}	LSB	00_{HEX}	RO	Maximum current value measured in configured measurement sequence
	14_{HEX}	MSB	80_{HEX}	RO	
adcCmin	15_{HEX}	LSB	FF_{HEX}	RO	Minimum current value measured in configured measurement sequence
	16_{HEX}	MSB	$7F_{HEX}$	RO	
adcVmax	17_{HEX}	LSB	00_{HEX}	RO	Maximum voltage value measured in configured measurement sequence
	18_{HEX}	MSB	80_{HEX}	RO	
adcVmin	19_{HEX}	LSB	FF_{HEX}	RO	Minimum voltage value measured in configured measurement sequence
	$1A_{HEX}$	MSB	$7F_{HEX}$	RO	
adcCrcv	$1B_{HEX}$	LSB	00_{HEX}	RO	Counter register containing the number of current measurements
	$1C_{HEX}$	MSB	00_{HEX}	RO	
adcCtcv	$1D_{HEX}$	---	00_{HEX}	RO	Counter register containing the number of current measurements greater than or equal to the threshold
adcVrcv	$1E_{HEX}$	---	00_{HEX}	RO	Counter register containing the number of voltage measurements
Unused	$1F_{HEX}$	---	00_{HEX}	---	---

Name	Address	Order	Default	Access	Short Description
sleepTCurCnt	20_{HEX}	LSB	00_{HEX}	RO	Current sleep timer value
	21_{HEX}	MSB	00_{HEX}	RO	
Unused	22_{HEX} - $2F_{HEX}$	---	00_{HEX}	---	---
adcCgan	30_{HEX}	LSB	00_{HEX}	RW	Digital gain correction for current channel
	31_{HEX}	---	00_{HEX}	RW	
	32_{HEX}	MSB	80_{HEX}	RW	
adcCoff	33_{HEX}	LSB	00_{HEX}	RW	Digital offset correction for current channel
	34_{HEX}	---	00_{HEX}	RW	
	35_{HEX}	MSB	00_{HEX}	RW	
adcVgan	36_{HEX}	LSB	00_{HEX}	RW	Digital gain correction for voltage channel
	37_{HEX}	---	00_{HEX}	RW	
	38_{HEX}	MSB	80_{HEX}	RW	
adcVoff	39_{HEX}	LSB	00_{HEX}	RW	Digital offset correction for voltage channel
	$3A_{HEX}$	---	00_{HEX}	RW	
	$3B_{HEX}$	MSB	00_{HEX}	RW	
adcTgan	$3C_{HEX}$	LSB	00_{HEX}	RW	Digital gain correction for temperature channel
	$3D_{HEX}$	MSB	80_{HEX}	RW	
adcToff	$3E_{HEX}$	LSB	00_{HEX}	RW	Digital offset correction for temperature channel
	$3F_{HEX}$	MSB	00_{HEX}	RW	
adcCrcl	40_{HEX}	LSB	00_{HEX}	RW	Number of current measurements before ready strobe is generated
	41_{HEX}	MSB	00_{HEX}	RW	
adcCrth	42_{HEX}	LSB	00_{HEX}	RW	Absolute current value is compared to this threshold in Current Threshold Comparator Mode
	43_{HEX}	MSB	00_{HEX}	RW	
adcCtcl	44_{HEX}	---	00_{HEX}	RW	Number of current measurements greater than or equal to the threshold before “set interrupt” strobe is generated
adcVrcl	45_{HEX}	---	00_{HEX}	RW	Number of voltage measurements before ready strobe is generated
adcVth	46_{HEX}	LSB	00_{HEX}	RW	Voltage threshold level for Threshold Comparator (unsigned) or Accumulator (signed) Modes
	47_{HEX}	MSB	00_{HEX}	RW	
adcCaccth	48_{HEX}	LSB	00_{HEX}	RW	Accumulator threshold for current channel
	49_{HEX}	---	00_{HEX}	RW	
	$4A_{HEX}$	---	00_{HEX}	RW	
	$4B_{HEX}$	MSB	00_{HEX}	RW	
adcTmax	$4C_{HEX}$	---	00_{HEX}	RW	Upper threshold for temperature measurement
adcTmin	$4D_{HEX}$	---	00_{HEX}	RW	Lower threshold for temperature measurement
adcAcmp	$4E_{HEX}$	LSB	30_{HEX}	RW	ADC function enable register
	$4F_{HEX}$	MSB	00_{HEX}	RW	
adcGomd	50_{HEX}	---	10_{HEX}	RW	Reference voltage and sigma-delta modulator (SDM) configuration
adcSamp	51_{HEX}	---	00_{HEX}	RW	Oversampling and filter configuration
adcGain	52_{HEX}	---	00_{HEX}	RW	Control register for analog amplifiers
pwrCfgFp	53_{HEX}	---	00_{HEX}	RW	Power configuration register for full-power state
irqEna	54_{HEX}	LSB	00_{HEX}	RW	Interrupt enable register
	55_{HEX}	MSB	00_{HEX}	RW	
adcCtrl	56_{HEX}	---	00_{HEX}	RW	ADC control register for full-power (FP) state
adcPoCoGain	57_{HEX}	---	00_{HEX}	RW	Post-correction gain configuration
Unused	58_{HEX} - $5E_{HEX}$	---	00_{HEX}	---	---
discCvtCnt	$5F_{HEX}$	---	00_{HEX}	RW	Configuration register for some of the power-down states
sleepTAdcCmp	60_{HEX}	LSB	00_{HEX}	RW	Compare value for ADC trigger timer
	61_{HEX}	MSB	00_{HEX}	RW	

© 2016 Integrated Device Technology, Inc.

Name	Address	Order	Default	Access	Short Description
sleepTCmp	62_{HEX}	LSB	00_{HEX}	RW	Compare value for sleep timer
	63_{HEX}	MSB	00_{HEX}	RW	
pwrCfgLp	64_{HEX}	---	20_{HEX}	RW	Power configuration register for power-down modes
gotoPd	65_{HEX}	---	00_{HEX}	WO	Power-down entrance register
Unused	66_{HEX} - 67_{HEX}	---	00_{HEX}	---	---
cmdExe	68_{HEX}	---	02_{HEX}	WO / RW	Command execution register
Unused	69_{HEX} - $6F_{HEX}$	---	00_{HEX}	---	---
wdogCnt	70_{HEX}	LSB	FF_{HEX}	RO	Current watchdog counter value
	71_{HEX}	MSB	FF_{HEX}	RO	
wdogPresetVal	72_{HEX}	LSB	FF_{HEX}	RW	Preset value for watchdog counter
	73_{HEX}	MSB	FF_{HEX}	RW	
wdogCfg	74_{HEX}	---	09_{HEX}	RW	Configuration register for watchdog counter
Unused	75_{HEX} - 77_{HEX}	---	00_{HEX}	---	---
lpOscTrimCnt	78_{HEX}	LSB	00_{HEX}	RO	Result counter of low-power oscillator trim circuit
	79_{HEX}	MSB	00_{HEX}	RO	
irefLpOsc	$7A_{HEX}$	---	52_{HEX}	RW	Trim value for low-power oscillator
lpOscTrim	$7B_{HEX}$	---	04_{HEX}	RW	Configuration register for trim circuit of low-power oscillator
Unused	$7C_{HEX}$ - $7F_{HEX}$	---	00_{HEX}	---	---
swRst	80_{HEX}	---	00_{HEX}	WO	Software reset
Unused	81_{HEX} - AF_{HEX}	---	00_{HEX}	---	---
sdmClkCfgLp	$B0_{HEX}$	LSB	18_{HEX}	RW	Clock configuration for SDM clock in power-down state
	$B1_{HEX}$	MSB	00_{HEX}	RW	
sdmClkCfgFp	$B2_{HEX}$	LSB	08_{HEX}	RW	Clock configuration for SDM clock in full-power state
	$B3_{HEX}$	MSB	90_{HEX}	RW	
linCfg	$B4_{HEX}$	---	00_{HEX}	RW / W1C	Configuration register for LIN control logic
linShortFilter	$B5_{HEX}$	---	$0F_{HEX}$	RW	Configuration for LIN short de-bounce filter
linShortDelay	$B6_{HEX}$	---	$4F_{HEX}$	RW	Configuration for LIN short TX-RX delay
linWuDelay	$B7_{HEX}$	---	14_{HEX}	RW	Configuration for LIN wake-up time
pullResEna	$B8_{HEX}$	---	FF_{HEX}	RW	Configuration register for pull-down resistors
funcDis	$B9_{HEX}$	---	00_{HEX}	RW	Disable bits for dedicated functions
versionCode	BA_{HEX}	LSB	00_{HEX}	RO	Version code
	BB_{HEX}	MSB	03_{HEX}	RO	
Unused	BC_{HEX} - BF_{HEX}	---	00_{HEX}	---	---
pwrTrim	$C0_{HEX}$	---	$7C_{HEX}$	RW	Trim bits for voltage regulators and bandgap
irefOsc	$C1_{HEX}$	LSB	10_{HEX}	RW	Trim values for high-precision oscillator
	$C2_{HEX}$	MSB	40_{HEX}	RW	
ibiasLinTrim	$C3_{HEX}$	---	10_{HEX}	RW	Bias current trim register for LIN block
Reserved	$C4_{HEX}$	---	00_{HEX}	RW	---
Reserved	$C5_{HEX}$	---	00_{HEX}	RW	---
Reserved	$C6_{HEX}$	---	00_{HEX}	RW	---
Reserved	$C7_{HEX}$	---	00_{HEX}	RW	---
Reserved	$C8_{HEX}$	---	00_{HEX}	RW	---
Reserved	$C9_{HEX}$	---	00_{HEX}	RW	---
Reserved	CA_{HEX}	---	08_{HEX}	RW	---
Reserved	CB_{HEX}	---	00_{HEX}	RW	---

© 2016 Integrated Device Technology, Inc.

Name	Address	Order	Default	Access	Short Description
Reserved	CC_{HEX}	---	00_{HEX}	RW	---
Reserved	CD_{HEX}	---	00_{HEX}	RW	---
Reserved	CE_{HEX}	---	00_{HEX}	RW	---
Reserved	CF_{HEX}	---	00_{HEX}	RW	---
adcChan	$D0_{HEX}$	---	00_{HEX}	RW	Analog multiplexer configuration during test/ diagnosis
adcDiag	$D1_{HEX}$	---	80_{HEX}	RW	Enable register for test/diagnosis
currentSrcEna	$D2_{HEX}$	---	00_{HEX}	RW	Enable register for current sources
Reserved	$D3_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D4_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D5_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D6_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D7_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D8_{HEX}$	---	00_{HEX}	RW	---
Reserved	$D9_{HEX}$	---	00_{HEX}	RW	---
Reserved	DA_{HEX}	---	$B8_{HEX}$	RW	---
Reserved	DB_{HEX}	---	00_{HEX}	RW	---
Reserved	DC_{HEX}	---	00_{HEX}	RW	---
Reserved	DD_{HEX}	---	00_{HEX}	RW	---
Reserved	DE_{HEX}	---	00_{HEX}	RW	---
Reserved	DF_{HEX}	---	00_{HEX}	RW	---
OTP	$E0_{HEX}$ - FF_{HEX}	---	---	RO	OTP raw data

3.3. SBC Clock and Reset Logic

3.3.1. Clocks

The SBC contains two different oscillators, a low-power oscillator (LP oscillator) providing a clock of 125kHz with an accuracy of +/- 3% and a high-precision oscillator (HP oscillator) providing a clock of 20MHz with an accuracy of +/- 1%. The low-power oscillator is always active except in the OFF state while the high-precision oscillator is only active in the SBC's full-power (FP) state. The clock from the high-precision oscillator is routed to the MCU via the internal MCU_CLK node (see Figure 2.3).

Three different internal clocks are generated from the two clocks from the oscillators for the digital core of the SBC:

- **Low-power clock (lpClk):** This clock is directly driven by the low-power oscillator and has a frequency of 125kHz. It is used for the watchdog timer, the sleep timer, and the power management unit.
- **Divided clock (divClk):** This clock is derived from the high-precision oscillator and has a frequency of 4MHz. It is used for the register file, the low-power oscillator trimming circuit, the LIN support logic, and the OTP controller.
- **Multiplexed clock (muxClk):** This clock is identically to the divClk in the full-power (FP) state and identically to lpClk in the LP and ULP states. It is used for the ADC controller unit and the interrupt controller.

Both oscillators are trimmed during the production test, and the trim values are stored in the OTP memory (`IREF_OSC_0`, `IREF_OSC_1`, `IREF_OSC_2`, `IREF_OSC_3`, `IREF_LP_OSC`; see Table 3.67). As it is important for correct operation of the MCU that the clock from the high-precision oscillator has the correct frequency, the two trimming values for the high-precision oscillator are protected by redundancy in the OTP. Software can check the validity of the trim values and the redundancy bits by reading the OTP raw data directly from the OTP via the SPI.

Note: The trimming values for both oscillators should also be stored in the MCU so that software is able to check the validity of the trimming values. On detection of errors in the OTP, software can write the correct values via SPI.

3.3.2. Trimming the Low-Power Oscillator

As the clock from the low-power oscillator is less accurate than the clock from the high-precision oscillator, a trimming circuit is implemented that trims the low-power oscillator using the divided clock divClk. There are two options for trimming the low-power oscillator. One option is to allow the hardware to update the trim value for the low-power oscillator automatically so that no user interference is necessary. For this, the user only needs to set the `lpOscTrimEna` and `lpOscTrimUpd` bits to 1 as well as setting the `lpOscTrimCfg` field as needed (see Table 3.4). The latter configuration value defines how many low-power clock periods are used for frequency calculation. While the trimming circuit is faster when fewer periods are used, the result of the frequency calculation is more accurate when more periods are used. In the first part of the trimming loop, the circuit determines the frequency of the low-power oscillator. When the measured frequency is too low, the hardware increments the trim value by 1; if it is too high, the hardware decrements the trim value by 1. Otherwise, the trim value remains unchanged. After changing the trim value, the hardware measures the (new) frequency. This algorithm is only stopped when software clears the `lpOscTrimEna` bit (trimming logic stops after a final update) or when any low-power state is entered.

The second option is to use the trim circuit only to measure the frequency but to update the trim value by software. This can be preferable when the target frequency is not equal to 125 kHz. For this, the user only needs to set the `lpOscTrimEna` bit to 1 and set the `lpOscTrimUpd` bit to 0 as well as setting the `lpOscTrimCfg` field as needed. Next, the user must clear the `lpOscTrimEna` bit without changing the other values in the register and must wait until the hardware has finished before calculating the frequency (wait until `SSW[6]` is 0). By reading the `lpOscTrimCnt` register, the user can calculate the actual frequency of the low-power oscillator using the following formula:

$$f_{LP} = \frac{f_{HP}}{\mathrm{lpOscTrimcnt} + 1} \cdot 2^{\mathrm{lpOscTrimCfg}+2} \qquad f_{HP} = 4\,\mathrm{MHz} \tag{1}$$

After determining the actual frequency, the user can change the trim value for the low-power oscillator `lpOscTrimVal` as required (see Table 3.3) and re-enable the trimming circuit to check the new frequency.

Note: The trimming circuit can be kept active when going to any low-power state. The PMU interrupts the trimming circuit on transition to the low-power state and restarts it after wakeup. This is needed as divClk is stopped in any low-power state.

3.3.3. Clock Trimming and Configuration Registers

3.3.3.1. Register "irefOsc" – Trim Values for the High-Precision Oscillator

Table 3.2 Register `irefOsc`

Name	Address	Bits	Default	Access	Description
irefTcOscTrim	$C1_{HEX}$	[4:0]	10000_{BIN}	RW	Trim value to minimize the temperature coefficient of the high-precision oscillator. **Note:** This value is automatically updated by the OTP controller after an SBC reset.
Unused		[6:5]	00_{BIN}	RO	Unused; always write as 0.
irefOscTrim[0]		[7]	0_{BIN}	RW	Trim value for the high-precision oscillator. The frequency of the high-precision oscillator increases (decreases) when this value is incremented (decremented). **Note:** This value is automatically updated by the OTP controller after an SBC reset.
irefOscTrim[8:1]	$C2_{HEX}$	[7:0]	40_{HEX}	RW	

3.3.3.2. Register "irefLpOsc" – Trim Value for the Low-power Oscillator

Table 3.3 Register `irefLpOsc`

Name	Address	Bits	Default	Access	Description
lpOscTrimVal	$7A_{HEX}$	[6:0]	$101\ 0010_{BIN}$	RW	Trim value for the low-power oscillator. The frequency of the low-power oscillator increases (decreases) when this value is incremented (decremented). **Note:** This value is automatically updated by the OTP controller after an SBC reset.
Unused		[7]	0_{BIN}	RO	Unused; always write as 0.

3.3.3.3. Register "lpOscTrim" – Configuration Register for the Low-Power Oscillator Trimming Circuit

Table 3.4 Register `lpOscTrim`

Name	Address	Bits	Default	Access	Description
lpOscTrimEna	$7B_{HEX}$	[0]	0_{BIN}	RW	If set to 1, enables the low-power oscillator trimming circuit. **Note:** When the user disables the trimming feature, the trimming logic continues its operation until it has finished the current calculation and then stops. The user can check that the trimming circuit has stopped by evaluating SSW[6], which is 0 when the trimming circuit is inactive.
lpOscTrimUpd		[1]	0_{BIN}	RW	Update bit for the low-power oscillator trimming circuit. When set to 1, the trimming circuit is allowed to update register `lpOscTrimVal`. When set to 0, no hardware update is performed. **Note:** Do not change while trimming circuit is active.
lpOscTrimCfg		[3:2]	01_{BIN}	RW	This value selects the number of clock periods of the low-power oscillator to be used to determine the frequency. 0 – 4 clock periods 1 – 8 clock periods 2 – 16 clock periods 3 – 32 clock periods **Note:** Do not change while trimming circuit is active.
Unused		[7:3]	00000_{BIN}	RO	Unused; always write as 0.

3.3.3.4. Register "lpOscTrimCnt" – Result Counter of the Low-Power Oscillator Trimming Circuit

Table 3.5 Register `lpOscTrimCnt`

Name	Address	Bits	Default	Access	Description
lpOscTrimCnt[7:0]	78_{HEX}	[7:0]	00_{HEX}	RO	Result counter of the low-power oscillator trimming circuit. This value will only be read when the trimming circuit is inactive (SSW[6] == 0).
lpOscTrimCnt[10:8]	79_{HEX}	[2:0]	000_{BIN}	RO	
Unused		[7:3]	00000_{BIN}	RO	Unused; always write as 0.

3.3.4. Resets

The main reset source is the integrated power-on-reset cell, which resets the complete digital core of the SBC when VDDE drops below 3.0V (typical). There are three other reset sources that reset the complete digital core of the SBC, except the watchdog timer and its configuration registers.

These additional reset sources are

- **Watchdog reset:** This reset occurs when the active watchdog timer expires without being handled by software.
- **Software reset:** This reset can be generated by the user by writing the value $A9_{HEX}$ to register `swRst`.
- **PMU error reset:** This reset occurs if the power management unit goes into an invalid state (e.g., due to cosmic radiation).

If any of these four resets occurs, the power-on procedure is executed, which powers up the required analog blocks and starts the download procedure for the OTP. This download procedure transfers the OTP contents into the appropriate registers if the OTP content is valid. The MCU_RSTN pin is also driven low, resetting the connected MCU. The MCU reset is released after the power-up procedure has finished.

The MCU is also reset when the system goes to OFF or ULP state because the power supply of the MCU is disabled in these power-down states. In this case, the MCU reset is released after a wake-up event has occurred and the MCU power supplies have stabilized.

Another possible reset source for the MCU is VddpReset, which is also generated by the power-on-reset cell when VDDE drops below 4.05V (typical). In this case, it cannot be guaranteed that VDDP, which is needed for correct operation of the MCU, is still valid if VDDP is trimmed to a high level of 3.3V.

The digital core of the SBC observes the input from the power-on-reset cell and generates the MCU reset only when all of the following conditions are true:

- VDDP is trimmed to 3.3V (bit `vddpTrim` of register `pwrTrim` set to 1).
- The ZSSC1956 system is in the full-power (FP) state, and the MCU is clocked.
- VDDP reset is not disabled (bit `disVddpRst` of register `funcDis` is set to 0; see Table 3.8).

3.3.4.1. The Reset Status

The MCU can easily check the reason for being reset by a single byte transfer to the SBC (SPI address byte only) and evaluating `SSW[10:8]` which contains the reason for the last reset (reset status). This value can be evaluated by the software for different actions after reset:

Reset status 0: In this case, the reset was generated by the power-on-reset cell. Both SBC and MCU are reset completely.

Reset status 1: The watchdog timer was not handled and has expired (see section 3.4). The MCU and all SBC logic are reset, except the watchdog timer and its configuration registers.

Reset status 2: Only the MCU is reset due to a wakeup from the ULP or OFF state.

Reset status 3: The software has forced a reset. The MCU and all SBC logic are reset, except the watchdog timer and its configuration registers.

Reset status 4: VDDP has dropped below 3.3V, and MCU was active. Only the MCU is reset.

Reset status 5: The PMU is in an illegal state. The MCU and all SBC logic are reset, except the watchdog timer and its configuration registers.

3.3.4.2. The Low-Voltage Flag

The low-voltage flag is part of the analog block. After power-on-reset, it has a low level. It can be set by software by writing the value 1 to bit `lvfSet` in register `cmdExe`. It is cleared by the power-on-reset cell when VDDE drops below 1.9V (typical). When VDDE drops below this threshold, it cannot be guaranteed that VDDL, which is used as the power supply for the RAM in the MCU, was high enough to guarantee the validity of the RAM contents. The low-voltage flag is mapped to `SSW[11]` where software can read its value.

3.3.4.3. Register "swRst" – Software Reset

Table 3.6 Register `swRst`

Name	Address	Bits	Default	Access	Description
swRst	80_{HEX}	[7:0]	00_{HEX}	WO	Writing $A9_{HEX}$ to this register forces a software reset, which resets the MCU as well as the SBC digital core except the watchdog timer and its configuration registers. Always reads as 0.

3.3.4.4. Register "cmdExe" – Triggering Command Execution by Software

Table 3.7 Register `cmdExe`

Name	Address	Bits	Default	Access	Description
wdogClr	68_{HEX}	[0]	0_{BIN}	RW	Writing 1 to this bit clears the watchdog timer. This bit is cleared by hardware after the watchdog is cleared. As long as the clear procedure is active, any further writes to this bit are rejected.
otpDownload		[1]	1_{BIN}	WO	Strobe register; write 1 to start the download procedure from the OTP; always reads as 0.
lvfSet		[2]	0_{BIN}	WO	Strobe register; write 1 to set the low-voltage flag; always reads as 0.
Unused		[7:3]	00000_{BIN}	RO	Unused; always write as 0.

3.3.4.5. Register "funcDis" – Disabling VDDP Reset and STO Output Pin

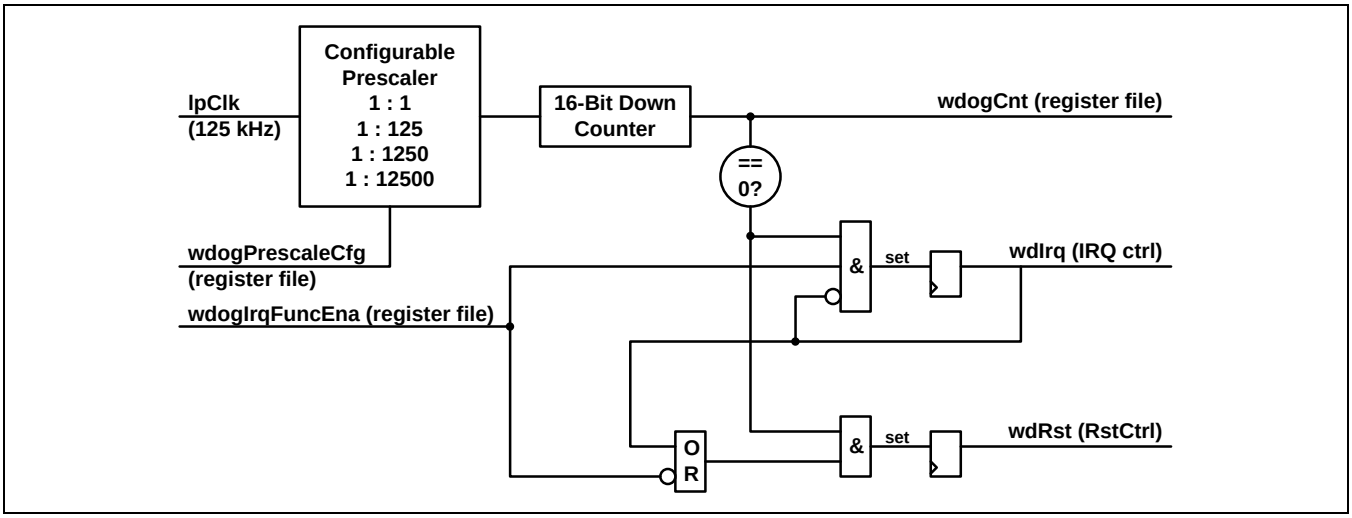
Table 3.8 Register `funcDis`

Name	Address	Bits	Default	Access	Description
disVddpRst	$B9_{HEX}$	[0]	0_{BIN}	RW	When set to 1, VddpReset does not reset the MCU.
disStoOut		[1]	0_{BIN}	RW	When set to 1, the output driver of the STO pin is disabled.
Unused		[7:2]	000000_{BIN}	RO	Unused; always write as 0.

3.4. SBC Watchdog Timer

The SBC contains a configurable watchdog timer (down counter) for the ZSSC1956 when it is running using the clock from the low-power oscillator. It is used to recover from an invalid software or hardware state. To avoid a reset of the system, the watchdog must be periodically serviced. The only part of the system that will not be reset by the watchdog reset is the watchdog itself and its configuration registers.

Figure 3.2 Structure of the Watchdog Timer



By default, the watchdog timer is active starting with a counter value of $FFFF_{HEX}$ and a prescaler of 125. This is done to guarantee that the boot code of the MCU has enough time to finish. During the initialization phase of the system, software can disable, reconfigure, and restart the watchdog. Disabling the watchdog before configuration is required as all write accesses to the register `wdogPresetVal` and the register `wdogCfg` except the bits `wdogLock` and `wdogEna` (see Table 3.12) are blocked when the watchdog is active. As it takes multiple low-power clock cycles until the enable signal is evaluated in the watchdog clock domain, the `SSW[7]` bit (`wdActive`) must be checked to determine if write accesses are possible. To avoid any malfunction during reconfiguration, the prescaler bit fields are set to 0 and the counter register is set to $FFFF_{HEX}$ at disable.

When the watchdog is disabled, configuration is possible. The register `wdogPresetVal` contains the value that will be copied into the down counter in the first enable cycle or when the watchdog timer is serviced via the `wdogClr` bit in register `cmdExe`. The field `wdogPrescaleCfg` in register `wdogCfg` configures the prescaler. The resolution and maximum timeout for the watchdog depend on the configuration as shown in Table 3.9.

Table 3.9 Resolution and Maximum Timeout for Prescaler Configurations

`wdogPrescaleCfg` Setting	Prescaler Configuration	Resolution	Maximum Timeout
0	1:1	8 µs	524 ms
1	1:125	1 ms	65.5 s
2	1:1250	10 ms	655.3 s
3	1:12500	100 ms	6553.5 s

As the maximum timeout value might still be too small for some applications, the user can use the wdogPmDis bit in register wdogCfg to select whether the watchdog timer will be halted during any power-down state (bit set to 1) or not (bit set to 0).

It is also possible to use the watchdog timer as a wake-up source. When the wdogIrqFuncEna bit in register wdogCfg is set to 1 and the down counter reaches 0, an interrupt is generated instead of a reset which would wake up the system and the down counter reloads the preset value and continues its operation. When the watchdog timer expires for a second time without service, the watchdog reset is generated. If the wdogIrqFuncEna bit is set to 0, the reset is already generated when the timer expires for the first time.

After reconfiguration, the watchdog timer is re-enabled. To avoid further (accidental) changes to the watchdog timer configuration registers, the user can set the wdogLock bit in register wdogCfg to 1. If this bit is set, all write accesses are blocked. The wdogLock bit can only be cleared by a power-on reset.

When a debugger is connected to the system (SBC input DBGEN is 1), the watchdog timer is immediately halted as handling of the watchdog via the debugger might be too slow; however, the watchdog timer can still be serviced via the wdogClr bit in register cmdExe (see Table 3.7).

To perform the required periodic servicing of the watchdog timer, the user must write the value 1 to the wdogClr bit in register cmdExe. To avoid any malfunction if the watchdog is serviced too often, any consecutive write accesses to the wdogClr bit are blocked until the first clear process has finished.

Important: The preset value programmed to the wdogPresetVal register must never be 00_{HEX} as this would immediately cause a reset forcing the system into a dead lock. It is strongly recommended that software checks the programmed reload value before re-enabling the watchdog.

Important: The preset value must not be too small. The user must take into account critical system timings including power-up times and flash programming/erasing times.

Note: The reconfiguration of the registers wdogPresetVal and wdogCfg, including bits wdogEna and wdogLock, can be performed in a single SPI burst write access.

3.4.1. Watchdog Registers

3.4.1.1. Register "wdogPresetVal" – Preset Value for the Watchdog Timer

Table 3.10 Register wdogPresetVal

Name	Address	Bits	Default	Access	Description
wdogPresetVal[7:0]	72_{HEX}	[7:0]	FF_{HEX}	RW	Lower byte of the preset value of the watchdog timer. This value is loaded into the lower byte of the watchdog counter when the watchdog is enabled or when the watchdog is cleared. **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0) and when the watchdog is inactive (SSW[7] == 0).
wdogPresetVal[15:8]	73_{HEX}	[7:0]	FF_{HEX}	RW	Upper byte of the preset value of the watchdog timer. This value is loaded into the upper byte of the watchdog counter when the watchdog is enabled or when the watchdog is cleared. **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0) and when the watchdog is inactive (SSW[7] == 0).

3.4.1.2. Register "wdogCnt" – Present Value of Watchdog Timer

Table 3.11 Register wdogCnt

Name	Address	Bits	Default	Access	Description
wdogCnt[7:0]	70_{HEX}	[7:0]	00_{HEX}	RO	Lower byte of present watchdog timer value.
wdogCnt[15:8]	71_{HEX}	[7:0]	00_{HEX}	RO	Upper byte of present watchdog timer value.

3.4.1.3. Register “wdogCfg” – Watchdog Timer Configuration Register

Table 3.12 Register wdogCfg

Name	Address	Bits	Default	Access	Description
wdogEna	74_{HEX}	[0]	1_{BIN}	RW	Global enable bit for the watchdog timer. **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0).
wdogPmDis		[1]	0_{BIN}	RW	When this bit is set to 1, PMU stops the watchdog during any power-down state. **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0).
wdogIrqFuncEna		[2]	0_{BIN}	RW	When this bit is set to 1, the watchdog reloads the preset value when expiring for the first time and generates an interrupt instead of a reset. A reset will always be generated when the watchdog timer expires for the second time. **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0) and when the watchdog is inactive (SSW[7] == 0).
wdogPrescaleCfg		[4:3]	01_{BIN}	RW	Prescaler configuration: 0 – No prescaler active 1 – Prescaler of 125 is active 2 – Prescaler of 1250 is active 3 – Prescaler of 12500 is active **Note:** This bit can only be written when the watchdog is not locked (wdogLock == 0) and when the watchdog is inactive (SSW[7] == 0). See Table 3.9 for timing details.
Unused		[6:5]	00_{BIN}	RO	Unused; always write as 0
wdogLock		7	0_{BIN}	RWS	When this bit is set to 1, all write accesses to the other bits of this register as well as to the wdogPresetVal registers are ignored. This bit can only be written to 1 and is only cleared by a power-on reset.

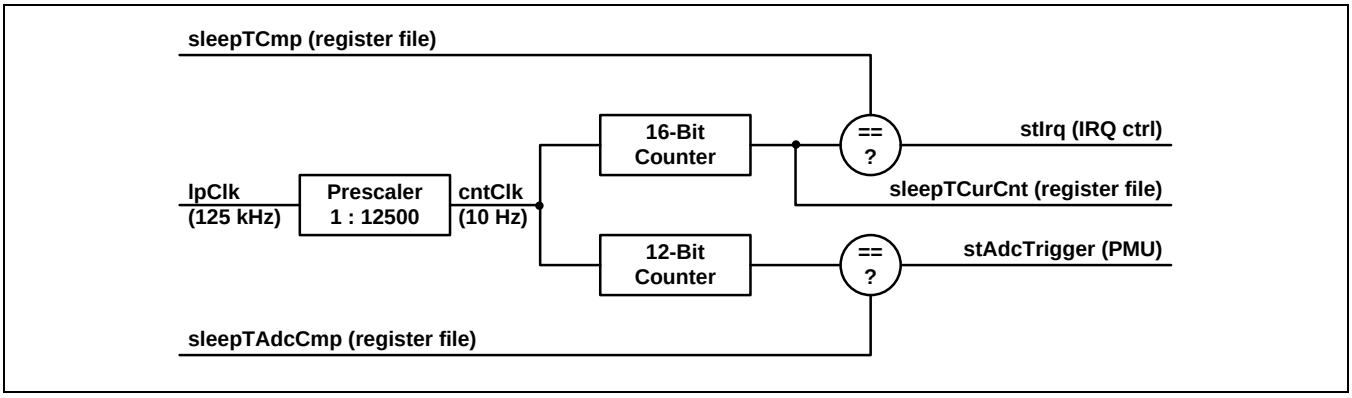
3.5. SBC Sleep Timer

The integrated sleep timer (up counter) is only active when the system is in any low-power state and it is running with the 125 kHz clock from the low-power oscillator.

The sleep timer consists of three blocks as illustrated in Figure 3.3:

- A fixed prescaler that divides the incoming 125 kHz clock from the low-power oscillator by 12500 to get a timer resolution of 10 Hz.
- A 16-bit counter that generates an interrupt (signal: stIrq) when the timer reaches the programmed compare value sleepTCmp (see Table 3.14).
- A 12-bit counter that triggers the PMU (with signal stAdcTrigger) when the timer reaches the programmed compare value sleepTAdcCmp to power-up the ADC blocks and to perform measurements if one of the discrete measurement scenarios are configured (see Table 3.13).

Figure 3.3 Structure of the Sleep Timer



When the system goes from full-power to any power-down state on request by the user, the prescaler and both counters are cleared and the 16-bit counter is enabled. Every 100 ms, triggered by the prescaler, the 16-bit counter is incremented until it reaches the programmed compare value `sleepTCmp`. When the compare value is reached, the timer stops and the interrupt controller is triggered to set the corresponding interrupt status flag (see section 3.6.1.1). The sleep timer is also stopped when the system returns to the full-power (FP) state. The user can determine the sleep duration by reading the register `sleepTCurCnt`, which returns the value of the 16-bit counter.

Note: Although the timer stops and the interrupt status bit is set when the compare value is reached, the system remains in the power-down state if the corresponding interrupt (bit 1 in `irqEna` register; see Table 3.17) is not enabled to drive the interrupt line IRQN.

Equation (2) can be used to determine the correct sleep time to be programmed. The sleep timer expires after 100ms for a compare value of 0, after 200ms for a compare value of 1, and so on.

$$\text{Sleep Time} = 100\text{ms} * (\text{sleepTCmp} + 1) \quad (2)$$

The 12-bit counter that triggers the PMU is only enabled during any power-down state when any discrete measurement scenario is configured. In this case, the counter is incremented each 100ms triggered by the prescaler. When the counter reaches the programmed compare value `sleepTAdcCmp`, a strobe for the PMU is generated and the 12-bit counter is reset to 0. Then it continues its operation. This counter is only stopped when the system returns to the full-power state, but it continues to operate when the sleep timer has expired if it was not enabled to wake up the system.

Equation (3) can be used to determine the correct ADC trigger time to be programmed. The ADC trigger timer expires after 100ms for a compare value of 0, after 200ms for a compare value of 1, and so on.

In general,

$$\text{ADC Trigger Time} = 100\text{ms} * (\text{sleepTAdcCmp} + 1) \quad (3)$$

Important: When both the sleep timer for wake-up and the ADC trigger timer for discrete measurements are used, special care must be taken when programming the compare values because when the sleep timer expires, the wake-up condition has higher priority over an active ADC measurement or an ADC trigger strobe.

3.5.1. Sleep Timer Registers

3.5.1.1. Register “sleepTAdcCmp” – Compare Value for ADC Trigger Timer

Table 3.13 Register sleepTAdcCmp

Name	Addr	Bits	Default	Access	Description
sleepTAdcCmp[7:0]	60_{HEX}	[7:0]	00_{HEX}	RW	Lower byte of compare value for the ADC trigger timer; the ADC trigger timer is only active if the system is in LP or ULP state and any discrete measurement scenario is configured generating periodic strobes for the PMU. ADC trigger time = 100 ms * (sleepTAdcCmp + 1)
sleepTAdcCmp[15:8]	61_{HEX}	[7:0]	00_{HEX}	RW	Upper byte of compare value for ADC trigger timer; ADC trigger timer is only active when the system is in LP or ULP state and any discrete measurement scenario is configured generating periodic strobes for the PMU. ADC trigger time = 100 ms * (sleepTAdcCmp + 1)

3.5.1.2. Register “sleepTCmp” – Compare Value for Sleep Timer

Table 3.14 Register sleepTCmp

Name	Addr	Bits	Default	Access	Description
sleepTCmp[7:0]	62_{HEX}	[7:0]	00_{HEX}	RW	Lower byte of compare value for sleep timer; sleep timer is only active if the system is in LP or ULP state. Sleep time = 100 ms * (sleepTCmp + 1)
sleepTCmp[15:8]	63_{HEX}	[7:0]	00_{HEX}	RW	Upper byte of compare value for sleep timer; sleep timer is only active when the system is in LP or ULP state. Sleep time = 100 ms * (sleepTCmp + 1)

3.5.1.3. Register “sleepTCurCnt” – Current Value of Sleep Timer

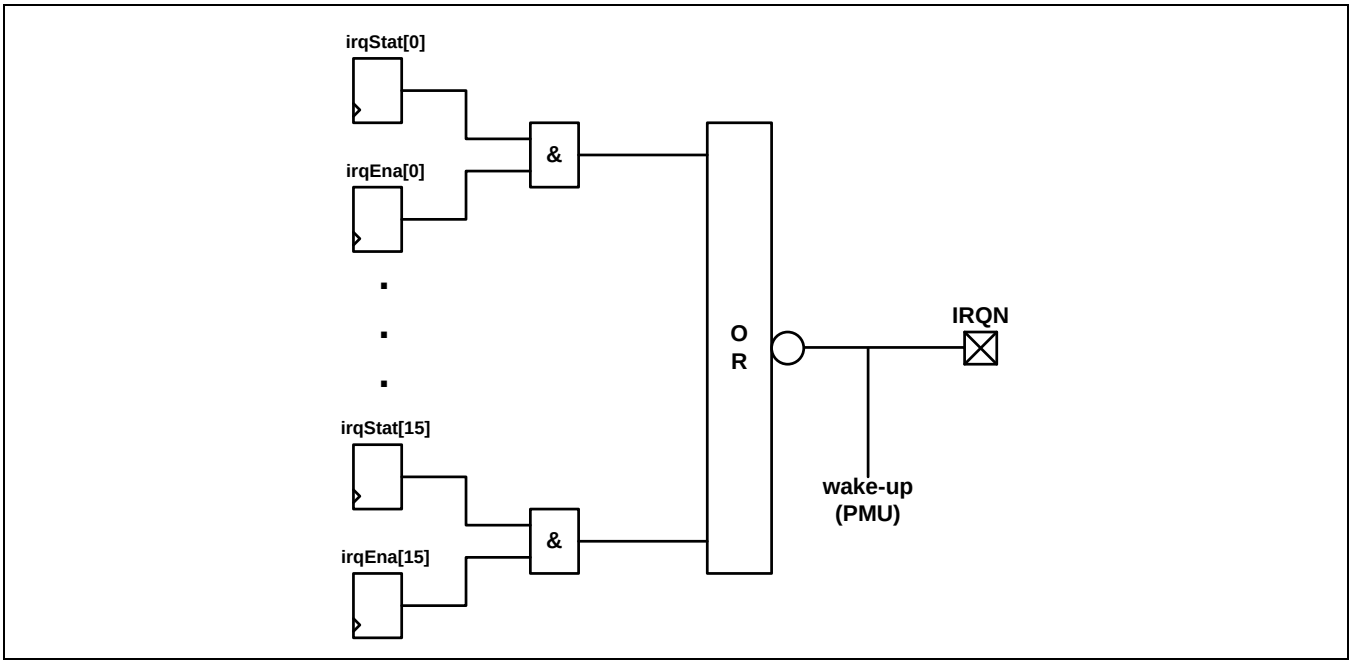
Table 3.15 Register `sleepTCurCnt`

Name	Addr	Bits	Default	Access	Description
sleepTCurCnt[7:0]	20_{HEX}	[7:0]	00_{HEX}	RO	Lower byte of the current sleep timer value. Since the timer is stopped in full-power (FP) state, the duration of the last power-down state can be determined: Sleep time = 100 ms * (sleepTCurCnt + 1) **Note:** value is only valid when SSW[1] (sleep timer valid stValid) is set.
sleepTCurCnt[15:8]	21_{HEX}	[7:0]	00_{HEX}	RO	Upper byte of the current sleep timer value. Since the timer is stopped in full-power (FP) state, the duration of the last power-down state can be determined: Sleep time = 100 ms * (sleepTCurCnt + 1) **Note:** value is only valid when SSW[1] (stValid) is set.

3.6. SBC Interrupt Controller

There are 16 different interrupt sources in the SBC system, each having a dedicated interrupt status bit in the `irqStat` register (Table 3.16) and a dedicated interrupt enable bit in the `irqEna` register (see Table 3.17). The interrupt controller captures each interrupt source in the interrupt status register independently of the interrupt enable settings. The interrupt controller combines all enabled interrupt status bits into the low-active interrupt signal that is used to drive the interrupt pin IRQN of the SBC and to wake up the system by the PMU. This means that interrupt status bits, which can always be set even when disabled, can only generate a wake-up event and drive the interrupt pin IRQN when they are enabled.

Figure 3.4 Generation of Interrupt and Wake-up



The user can determine the interrupt reason by reading the interrupt status register. The interrupt status register is cleared on each read access. Therefore software must ensure that it stores the read interrupt status value if needed to avoid loss of information.

3.6.1.1. Interrupt Sources

The bit mapping is the same for the interrupt enable register `irqEna` and the interrupt status register `irqStat`:

- **Bit 0:** **Watchdog Timer Interrupt**; status is set by the watchdog timer when the interrupt functionality of the watchdog timer is enabled and the watchdog timer expires for the first time.
- **Bit 1:** **Sleep Timer Interrupt**; status is set by the sleep timer when the sleep timer reaches the programmed compare value.
- **Bit 2:** **LIN TXD Timeout Interrupt**; status is set by the LIN support logic when the TXD input from the MCU is low for more than 10.24 ms.
- **Bit 3:** **LIN Short Interrupt**; status is set by the LIN support logic when a short is detected in the LIN PHY.
- **Bit 4:** **LIN Wakeup Interrupt**; status is set by the LIN support logic when a wake-up frame is detected on the LIN bus.
- **Bit 5:** **Current Conversion Result Ready Interrupt**; status is set by the ADC unit when a single current measurement (register `adcCrcl` == 0) or multiple current measurements defined by `adcCrcl` (register `adcCrcl` != 0) have been completed and the result is available.
- **Bit 6:** **Voltage Conversion Result Ready Interrupt**; status is set by the ADC unit when a single voltage measurement (register `adcVrcl` == 0) or multiple voltage measurements defined by the `adcVrcl` (register `adcVrcl` != 0) have been completed and the result is available.
- **Bit 7:** **Temperature Conversion Result Ready Interrupt**; status is set by the ADC unit when a single temperature measurement has been completed and the result is available.
- **Bit 8:** **Current Comparator Interrupt**; status is set by the ADC unit when the Current Threshold Counter Mode is enabled (register `adcAcmp[2:1]` != 0) and the absolute value of multiple (defined by register `adcCtcl`) current measurements exceeds the programmed current threshold (register `adcCrth`).
  Note: If the Current Threshold Counter Mode is enabled but `adcCtcl` is 0, this bit is always set independently of the threshold.
- **Bit 9:** **Voltage Comparator Interrupt**; status is set by the ADC unit if the `VThWuEna` bit (`adcAcmp[8]`) is set to 1 and a single measured voltage or the accumulated voltage measurements (depends on configured mode) drop below the programmed (register `adcVTh`) voltage threshold.
- **Bit 10:** **Temperature Threshold Interrupt**; status is set by the ADC unit when the `TWuEna` bit (`adcAcmp[10]`) is set to 1 and a temperature measurement is outside the specified temperature interval defined by registers `adcTmin` and `adcTmax`.
- **Bit 11:** **Current Accumulator Threshold Interrupt**; status is set by the ADC unit when the `CAccuThEna` bit (`adcAcmp[3]`) is set to 1 and the accumulated current values rise above the programmed threshold value (register `adcCaccTh`) for a positive threshold value or fall below the programmed threshold value for the negative threshold value.
- **Bit 12:** **Current Overflow Interrupt**; status is set by the ADC unit when the `COvrEna` bit (`adcAcmp[4]`) is set to 1 and the compensated value of a current measurement is outside of the representable range.
- **Bit 13:** **Voltage / Temperature Overflow Interrupt**; status is set by the ADC unit when the `VTOvrEna` bit (`adcAcmp[5]`) is set to 1 and the compensated value of a voltage or temperature measurement is outside of the representable range.

© 2016 Integrated Device Technology, Inc.

- **Bit 14:** **Current Over-Range Interrupt**; status is set by the ADC unit when the `COvrEna` bit (`adcAcmp[4]`) is set to 1 and the input from the current ADC is overdriven.
- **Bit 15:** **Voltage / Temperature Over-Range Interrupt**; status is set by the ADC unit when the `VTOvrEna` bit (`adcAcmp[5]`) is set to 1 and the input from the voltage / temperature ADC is overdriven.

3.6.1.2. Register "irqStat" – Interrupt Status Register

Table 3.16 Register `irqStat`

Name	Address	Bits	Default	Access	Description
irqStat[7:0]	00_{HEX}	[7:0]	00_{HEX}	RC	Lower byte of the interrupt status register as defined in section 3.6.1.1; each bit is set by hardware and cleared on read access.
irqStat[15:8]	01_{HEX}	[7:0]	00_{HEX}	RC	Upper byte of interrupt status register as defined in section 3.6.1.1; each bit is set by hardware and cleared on read access.

Note: To avoid loss of information, the hardware set condition has a higher priority than the read clear condition.

3.6.1.3. Register "irqEna" – Interrupt Enable Register

Table 3.17 Register `irqEna`

Name	Address	Bits	Default	Access	Description
irqEna[7:0]	54_{HEX}	[7:0]	00_{HEX}	RW	Lower byte of the interrupt enable register as defined in section 3.6.1.1; only enabled interrupts can drive the interrupt line and wake up the system; the bit mapping is the same as for the interrupt status register.
irqEna[15:8]	55_{HEX}	[7:0]	00_{HEX}	RW	Upper byte of the interrupt enable register as defined in section 3.6.1.1; only enabled interrupts can drive the interrupt line and wake up the system; the bit mapping is the same as for the interrupt status register.

Note: The interrupt enable bit for the LIN wake-up interrupt (`irqEna[4]`) is also used as the enable for the LIN wake-up frame detector within the PMU.

3.7. SBC Power Management Unit (PMU)

The power management unit (PMU) controls placing the SBC into the selected power down state, controlling the power down signals for the different analog blocks, and controlling the clocks for the digital logic. It also controls the other digital modules during the power-down state.

The system provides four different power states:

- FP (Full-Power State): In this state, all blocks are powered except the ADCs if software has not enabled them. All internal clocks are active (divClk and muxClk are 4MHz), and the MCU is also powered and clocked. When powered and enabled by software, the ADC clocks are generated from the clock from the high-precision oscillator.
- LP (Low-Power State): In this state, the high-precision oscillator and the LIN transmitter are powered down. The MCU clock is stopped, but the MCU remains powered. Depending on the selected measurement setup, the ADCs are also powered down during times of inactivity. Otherwise the ADC clocks are generated from the clock from the low-power oscillator.
- ULP (Ultra-Low-Power State): In this state, the high-precision oscillator and the LIN transmitter are powered down. The MCU clock is stopped, and the MCU is powered down. Depending on the selected measurement setup, the ADCs are also powered down during times of inactivity. Otherwise, the ADC clocks are generated from the clock from the low-power oscillator.
- OFF (Off State): In this state, all analog blocks except the digital power supply for the SBC and the RX part of the LIN PHY are powered down. The MCU clock is stopped, and the MCU is powered down.

To enter any of the power-down states (LP, ULP, or OFF), software must first set the `pdState` field of register `pwrCfgLp` to select the state and enable the interrupts needed as the wakeup source before writing $A9_{HEX}$ to register `gotoPd`. Immediately after $A9_{HEX}$ is written to the `gotoPd` register, the CSN line must be driven high. Although for all other register accesses, the CSN line can be kept low and the next SPI transfer can follow immediately, it is mandatory to drive CSN high for the power-down command. Otherwise, the PMU remains in the FP state.

Important: If no interrupt is enabled, the system can only be awakened by power-on-reset.

Important: The CSN line must be driven high to go to power-down after writing the value $A9_{HEX}$ to register `gotoPd`.

The following tasks are always performed on transition to any power-down state by the PMU:

- Both ADCs are stopped. Any active measurement is interrupted. ADC control is transferred to the PMU.
- The configuration values that can be configured independently for full-power and power-down states are switched to the power-down settings.
- The sleep timer is cleared and enabled.
- The MCU clock is stopped.
- The high-precision oscillator is powered down.
- The TX part of the LIN PHY is powered down.
- The source for the muxClk changes from divClk to lpClk.

When any of the enabled interrupts occurs and the interrupt pin IRQN is driven low, the system wakes up immediately; any ADC measurement that is active during power-down state is stopped. All mandatory blocks are powered up, and the system waits for stabilization before re-enabling the MCU clock.

If any of the enabled interrupts is already active on reception of the power-down command or becomes active on transition to the requested power-down state, the system rejects the power-down command or re-enables those blocks that are already powered down. Depending on the time when the power-down procedure was interrupted, it is possible that the sleep timer was not cleared. In this case, the sleep timer valid flag is cleared, signaling that the sleep timer value in register `sleepTCurCnt` is not valid. This flag is mapped to `SSW[1]`.

3.7.1. FP State

After the initial power-on reset when the OTP contents are downloaded into the registers and all blocks have stabilized, the system enters the FP state. In this state, all voltage regulators, both oscillators and the LIN PHY are powered but the ADCs are still powered down.

Important: Both ADCs are powered down after power-on reset.

To be able to use the ADCs, the user must first power up the required ADCs by programming register `pwrCfgFp`, bits `pwrAdcI` and/or `pwrAdcV`. The first bit enables the current ADC and the second bit enables the voltage/temperature ADC. In this register, there are three other bits that can be set by the user, but they should be handled with care as the system consumes less power when any of these bits is set but the accuracy of the measurement results is reduced:

- `lpEnaFp` when set to 1, the bias current for analog blocks is reduced to 10%
- `ulpEnaFp` when set to 1, the bias current for analog blocks is reduced to 5%
- `pdRefbufOcFp` when set to 1, the offset cancellation circuit in the reference buffer is powered down

Note: When both `lpEnaFp` and `ulpEnaFP` are set to 1, the bias current for analog blocks is reduced to 15%.

Important: These settings are only used in FP state. For configuration for the power-down states, register `pwrCfgLp` must be used.

The settings in register `pwrCfgFp` are preserved when entering any power-down state by executing the power-down command. The PMU overrides these settings or switches to the settings made in register `pwrCfgLp` on transition to power-down state. When the system wakes up and returns to the FP state, the PMU restores the settings as configured in `pwrCfgFp` regardless of whether any ADC was powered in power-down state or not.

3.7.2. LP and ULP States

The LP and ULP power-down states are used to save power while doing measurements with lower accuracy. In both states, the TX part of the LIN PHY and the high-precision oscillator are powered down and the MCU clock is stopped. The internal clock muxClk is driven by the low-power oscillator with a frequency of 125 kHz while the internal clock divClk is stopped. In ULP state, the two voltage regulators VDDP (IO voltage for SBC and MCU) and VDDC (core voltage for MCU) are powered down. In this case, the RAM_PROTN pad is driven low as the RAM in the MCU remains powered (connected to VDDL) and the signal RAM_PROTN is used to protect the RAM inputs. The state of the ADCs and the other analog blocks needed for measurements depends on the configured measurement setup for the power-down state (see following subsections). The blocks are powered when they are needed for measurement and powered down when they are not needed for measurement. This is controlled by the PMU as well as the control signals (*start, stop, mode*) for the digital ADC unit.

The main configuration register for the power-down behavior is register `pwrCfgLp`. The field `pdState` is used to select the power-down state to be entered on reception of the power-down command, and the field `pdMeas` is used to define the measurement setup to be used during the power-down state.

There are three other bits to configure the power-down behavior:

- `lpEnaLp` when set to 1, the bias current for analog blocks is reduced to 10%
- `ulpEnaLp` when set to 1, the bias current for analog blocks is reduced to 5%
- `pwrRefbufOcLp` when set to 1, the offset cancellation circuit in the reference buffer is powered up

Note: When both `lpEnaLp` and `ulpEnaLp` are set to 1, the bias current for analog blocks is reduced to 15%.

For the corresponding bits for the FP state, `lpEnaFp` and `ulpEnaFp`, the meaning is the same, but the default settings are different. While there is no bias current reduction during FP state (default setting for both bits is 0), the default bias current for the LP and ULP states is reduced to 10%. The meaning of the control bit for the offset cancellation differs: the FP state control bit is a power-down signal; the LP/ULP state control bit is a power-up bit. While the offset cancellation is enabled by default during the FP state (`pdRefbufOcFp == 0`), the offset cancellation is disabled by default during the LP or ULP state (`pwrRefbufOcLp == 0`). Both bits are configurable by the user.

On transition to the LP or ULP state, the sleep timer and the ADC trigger timer are cleared. While the sleep timer is always enabled during power-down states, the ADC trigger timer is only enabled when performing discrete measurements. When the sleep timer interrupt is enabled, the system wakes up when the sleep timer expires. If the sleep timer interrupt is not enabled, the sleep timer stops when it expires, but the ADC trigger timer, if enabled due to the measurement configuration, continues its operation. For wake-up, other interrupts must be enabled; e.g., LIN wakeup.

Note: The sleep timer is always active during LP and ULP state.

Note: When reading the sleep timer value after wake-up by another enabled interrupt, the sleep timer is only valid when it has not reached its compare value although the valid flag says valid. Whether the sleep timer is valid can be determined by the sleep timer status bit.

When the system wakes up and returns to FP state, the sleep timer is stopped. Software can read the sleep timer value to determine the duration of the power-down state.

3.7.2.1. Performing No Measurements during LP/ULP State

When the LP or ULP state has been entered, all analog blocks related to the ADCs are powered down. If the system goes to power down without performing any measurements, only three different wake-up sources are possible: the watchdog timer interrupt, the sleep timer interrupt, and the LIN wakeup interrupt.

Important: At least one of these interrupts must be enabled, as otherwise the system can only wake up by a power-on reset. If no interrupt is enabled, the system cannot wake up.

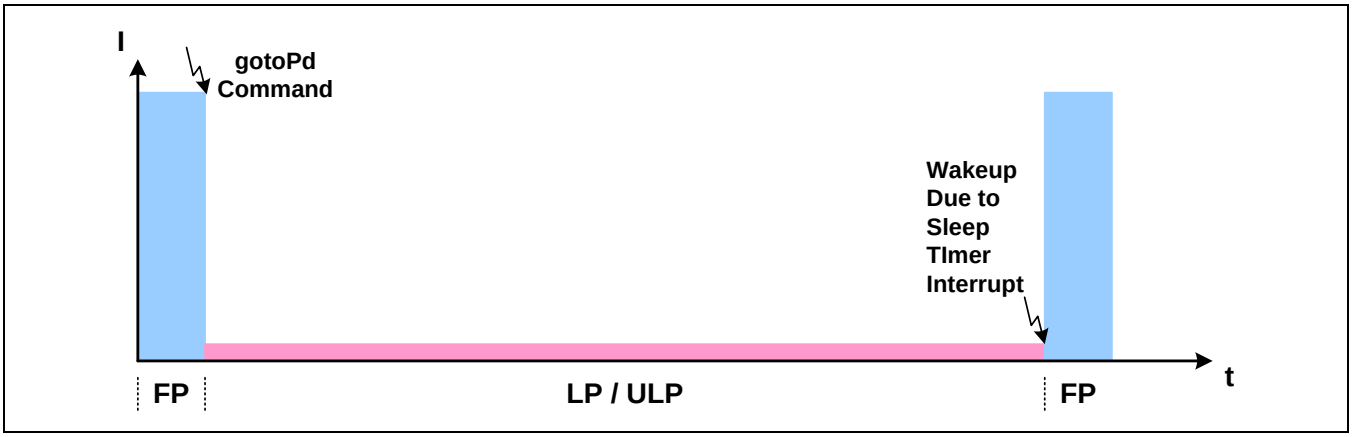
To go to LP or ULP state without performing measurements, the following tasks must be done:

- Enable at least one of the following interrupts:
  - Set `irqEna[0]` to 1 to enable the watchdog interrupt to wake up the system.
  - Set `irqEna[1]` to 1 to enable the sleep timer to wake up the system.
  - Set `irqEna[4]` to 1 to enable the LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
- Configure the sleep timer compare value (register `sleepTCmp`) if needed.
- Set `pdState` to 0 or 1 to configure the LP state or to 2 to configure the ULP state.
- Set `pdMeas` to 0 to configure the system to perform no measurements.
- Set `lpEnaLp`, `ulpEnaLp` and `pwrRefbufOcLp` as needed.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When an enabled interrupt occurs, the system wakes up and the settings from register `pwrCfgFp` are restored. When all blocks have stabilized, the MCU clock is re-enabled and if coming out of the ULP state, the MCU reset is released.

Figure 3.5 LP/ULP State without any Measurements

Note: The sleep timer interrupt is used as the wake-up source in this example, but it could also be the watchdog timer interrupt or the LIN wakeup interrupt.



3.7.2.2. Performing Discrete Measurements of Current during LP/ULP State

The system can be configured to periodically enable the current ADC and to measure the current during the LP or ULP state. The current ADC can be configured to perform several current measurements during each measurement phase (indicated by green blocks in Figure 3.6). Upon entering the LP/ULP state and between the measurements, the current ADC is powered down. The voltage/temperature ADC is powered down for the entire power-down period. The PMU powers up the current ADC when triggered by the ADC trigger timer. Possible wake-up sources during this scenario are the watchdog timer interrupt, the sleep timer interrupt, the LIN wakeup interrupt, or any of the ADC interrupts related to current.

Important: If no interrupt is enabled, the system cannot wake up.

To go to LP or ULP state and perform discrete current measurements, the following tasks must be done:

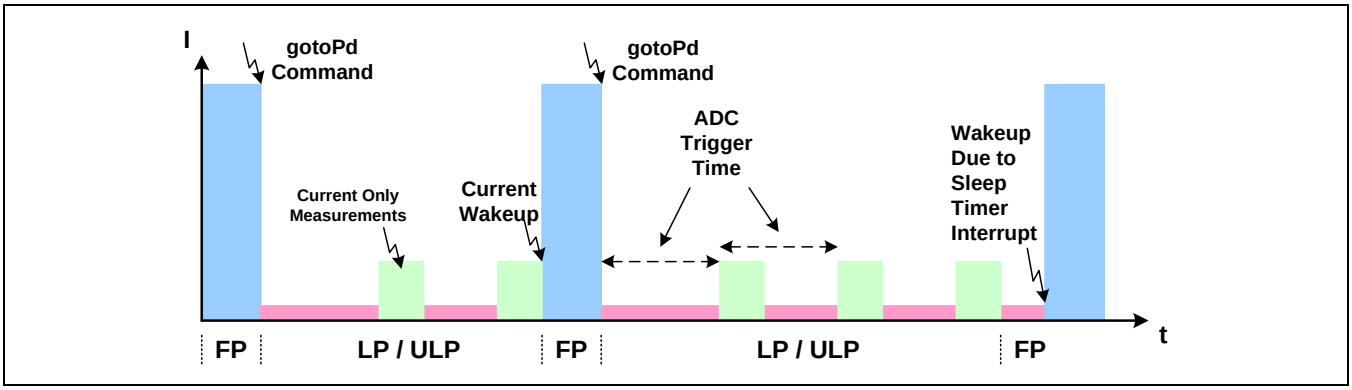
- Enable at least one of the following interrupts:
  - Set `irqEna[0]` to 1 to enable the watchdog interrupt to wake up the system.
  - Set `irqEna[1]` to 1 to enable the sleep timer to wake up the system.
  - Set `irqEna[4]` to 1 to enable the LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
  - Enable any ADC interrupt related to current.
- Configure the sleep timer compare value (register `sleepTCmp`) if needed.
- Configure the ADC trigger timer compare value (register `sleepTAdcCmp`) as needed.
- Set `pdState` to 0 or 1 to configure LP state or to 2 to configure ULP state.
- Set `pdMeas` to 1 to configure the system to perform discrete current measurements.
- Set `lpEnaLp`, `ulpEnaLp` and `pwrRefbufOcLp` as needed.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When an enabled interrupt occurs, the system wakes up and the settings from register `pwrCfgFp` are restored. When all blocks have stabilized, the MCU clock is re-enabled and if coming out of ULP state, the MCU reset is released.

Important: If any measurement is active while an enabled interrupt occurs (e.g., the sleep timer expires), the measurement is interrupted and the system returns to the FP state.

In example shown in Figure 3.6, the first wakeup is by the ADC and the second wake-up is by the sleep timer; however, the wakeups could be other combinations of the watchdog timer interrupt, sleep timer interrupt, and/or LIN wakeup interrupt.

Figure 3.6 LP/ULP State Performing Only Current Measurements



3.7.2.3. Performing Discrete Measurements of Current, Voltage, and Internal Temperature during LP/ULP State

The system can be configured to periodically enable both ADCs and to measure current, voltage, and internal temperature or external temperature (see section 3.7.2.4 for external temperature) during the LP or ULP state. The sequence can be selected in the `pdMeas` bit field [4:2] in register `pwrCfgLp`, which also selects whether internal or external temperature is measured. The period between each measurement is determined by the ADC trigger timer (`sleepTAdcCmp`). The current ADC can be configured to perform multiple current measurements during each measurement phase (indicated by green and orange blocks in Figure 3.7 to Figure 3.9) while the voltage/temperature ADC can be configured to perform multiple voltage measurements (orange blocks in Figure 3.7 to Figure 3.9). After performing the configured number of voltage measurements, the PMU changes the configuration for the voltage/temperature ADC and performs a single measurement of the internal temperature. Voltage and temperature are not measured in every loop if the ADCs are configured for measuring only current in a specified number of initial loops. The user can configure register `discCvtCnt` so that in the first `discCvtCnt` loops, only current is measured before voltage and temperature are measured in the next loop.

Upon entering the LP/ULP state and between the measurements, both ADCs are powered down. The PMU powers up the current ADC when triggered by the ADC trigger timer. The voltage/temperature ADC is only powered up after `discCvtCnt` current-only measurements have been performed. Possible wake-up sources in this setup are all interrupts except the LIN short and LIN TXD timeout interrupts.

Important: If no interrupt is enabled, the system cannot wake up.

To go to the LP or ULP state and perform measurements of discrete current, voltage, and internal temperature, the following tasks must be done:

- Enable at least one of the following interrupts:
  - Set `irqEna[0]` to 1 to enable the watchdog interrupt to wake up the system.
  - Set `irqEna[1]` to 1 to enable the sleep timer to wake up the system.
  - Set `irqEna[4]` to 1 to enable LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
  - Enable any ADC interrupt.
- Configure the sleep timer compare value (register `sleepTCmp`) if needed.
- Configure the ADC trigger timer compare value (register `sleepTAdcCmp`) as needed.
- Set `pdState` to 0 or 1 to configure the LP state or to 2 to configure the ULP state.
- Set `pdMeas` to 2 to configure the system to perform discrete current, voltage, and internal temperature measurements.
- Set `discCvtCnt` as needed. This register defines the number of current-only measurement loops before performing measurement of all three parameters.
- Set `lpEnaLp`, `ulpEnaLp` and `pwrRefbufOcLp` as needed.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When an enabled interrupt occurs, the system wakes up and the settings from register `pwrCfgFp` are restored. When all blocks have stabilized, the MCU clock is re-enabled and if coming out of the ULP state, the MCU reset is released.

Important: If any measurement is active while an enabled interrupt occurs (e.g., the sleep timer expires) the measurement is interrupted and the system returns to the FP the state.

Note: If register `discCvtCnt` is set to 0, voltage and temperature are measured in each loop (the default setting).

***Figure 3.7 LP/ULP State Performing Current, Voltage, and Temperature Measurements* (`discCvtCnt`==2)**

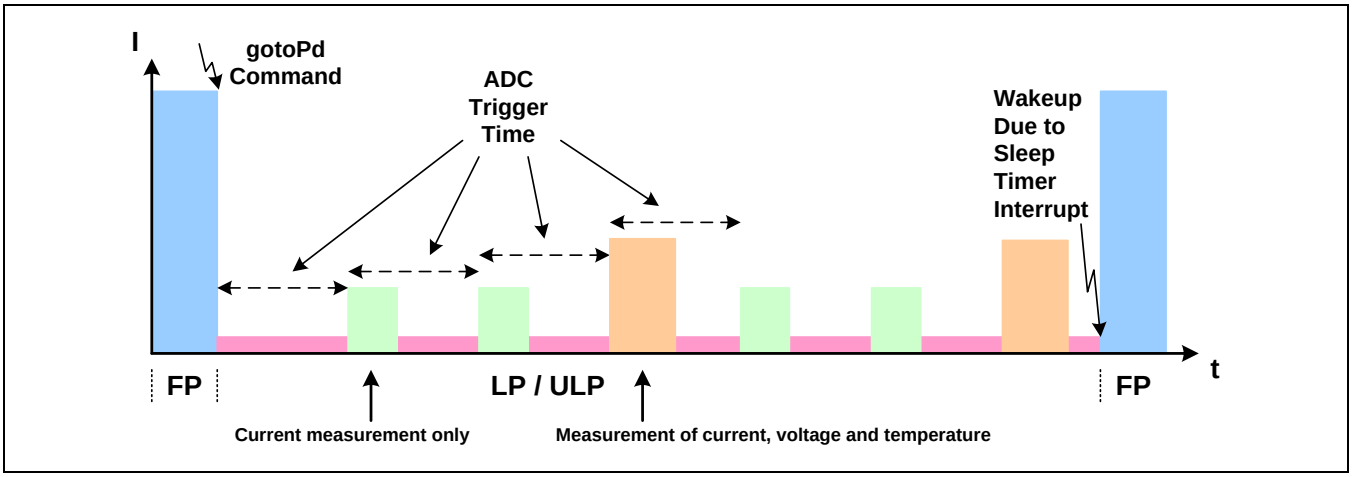


***Figure 3.8 LP/ULP State Performing Current, Voltage, and Temperature Measurements* (`discCvtCnt`==5)**

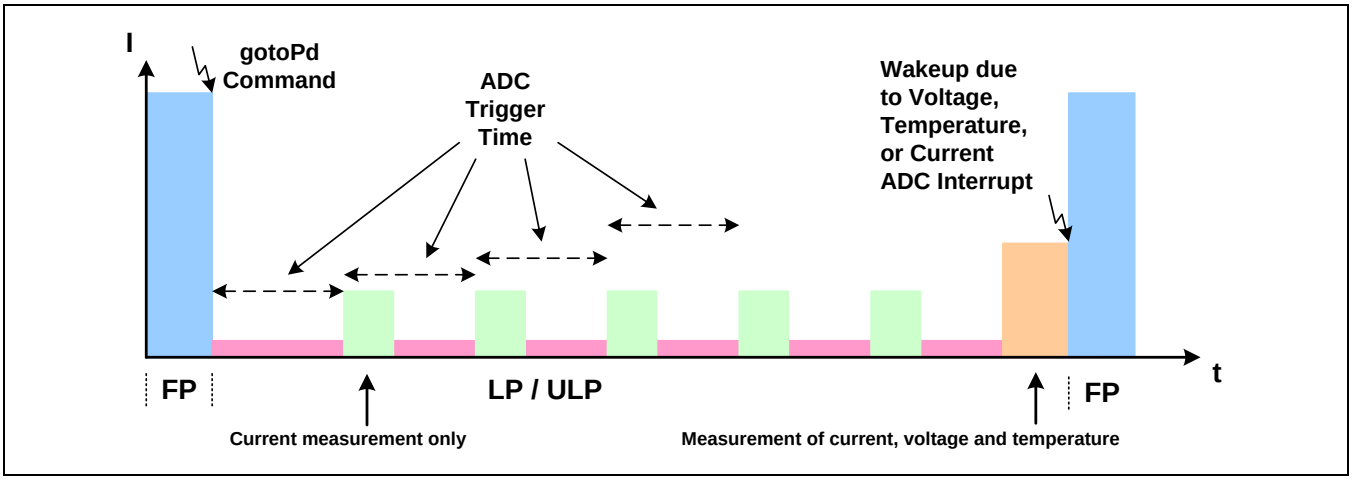
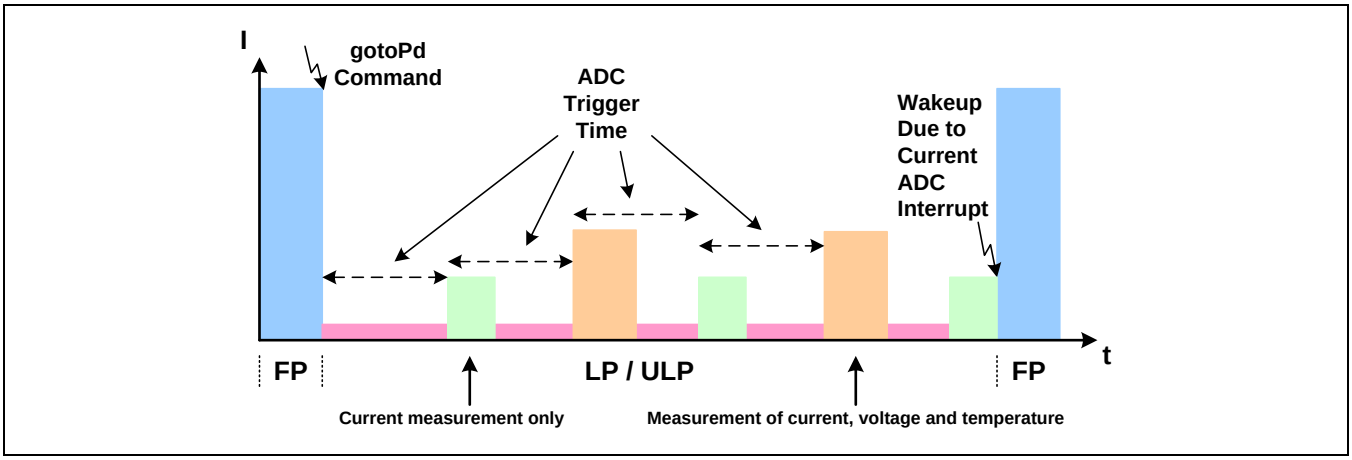


Figure 3.9 LP/ULP State Performing Current, Voltage, and Temperature Measurements (`discCvtCnt==1`)



3.7.2.4. Performing Discrete Measurements of Current, Voltage and External Temperature during LP/ULP State

This setup is the same as the configuration described in the previous section, except that the external instead of the internal temperature is measured. To use this option, `pdMeas` must be set to 3.

3.7.2.5. Performing Continuous Measurements of Current during LP/ULP State

The system can be configured to perform continuous current measurements during the LP or ULP state. While the current ADC is powered up during the entire power-down state, the voltage/temperature ADC is powered down.

The current ADC is powered up on entering the LP/ULP state if it was not already powered up during the FP state. The ADC trigger timer is not enabled as the measurement is continuous. Possible wake-up sources during this scenario are the watchdog timer interrupt, the sleep timer interrupt, the LIN wakeup interrupt, or any of the ADC interrupts related to current.

Important: If no interrupt is enabled, the system cannot wake up.

To go to the LP or ULP state and perform continuous current measurements, the following tasks must be done:

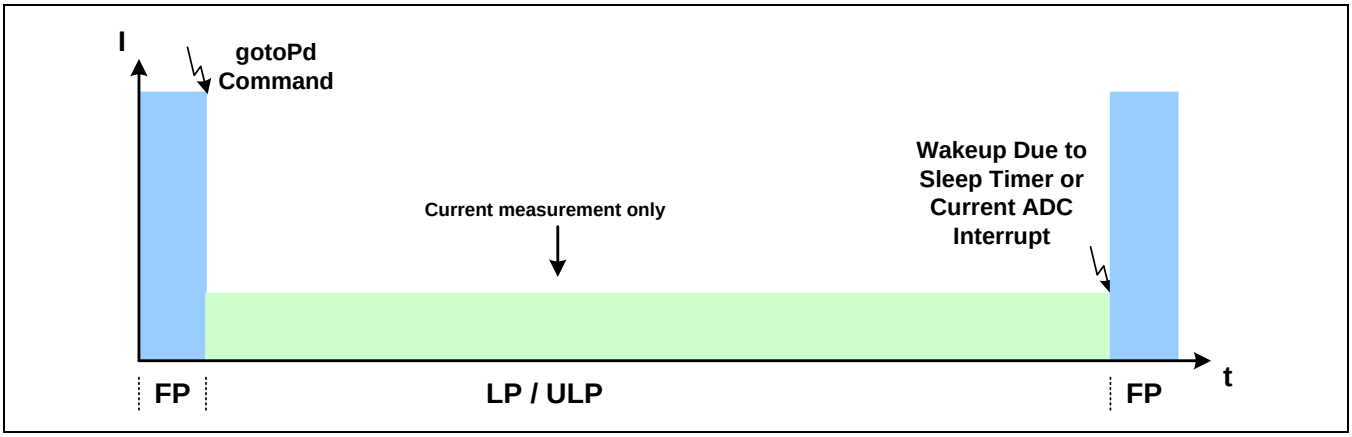
- Enable at least one of the following interrupts:
  - Set `irqEna[0]` to 1 to enable the watchdog interrupt to wake up the system.
  - Set `irqEna[1]` to 1 to enable the sleep timer to wake up the system.
  - Set `irqEna[4]` to 1 to enable the LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
  - Enable any ADC interrupt related to current.
- Set up the sleep timer compare value (register `sleepTCmp`) if needed.
- Set `pdState` to 0 or 1 to configure LP state or to 2 to configure ULP state.
- Set `pdMeas` to 4 to configure the system to perform continuous current measurements.
- Set `lpEnaLp`, `ulpEnaLp` and `pwrRefbufOcLp` as needed.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When an enabled interrupt occurs, the system wakes up and the settings from register `pwrCfgFp` are restored. When all blocks have stabilized, the MCU clock is re-enabled and if coming out of ULP state, the MCU reset is released.

Important: If any measurement is active while an enabled interrupt occurs (e.g., the sleep timer expires), the measurement is interrupted and the system returns to FP state.

Figure 3.10 LP/ULP State Performing Continuous Current-Only Measurements

Note: The sleep timer interrupt or an ADC interrupt related to current is used as the wake-up source in this example, but it could also be the watchdog timer interrupt or the LIN wakeup interrupt.



3.7.2.6. Performing Continuous Measurements of Current and Voltage during LP/ULP State

The system can be configured to perform continuous current and voltage measurements during the LP or ULP state. Both ADCs are powered up during the entire power-down state.

The ADCs are powered up on entering the LP/ULP state if they were not already powered up during the FP state. The ADC trigger timer is not enabled as the measurement is continuous. Possible wake-up sources during this scenario are the watchdog timer interrupt, the sleep timer interrupt, the LIN wakeup interrupt, or any of the ADC interrupts related to current or voltage.

Important: If no interrupt is enabled, the system cannot wake up.

To go to the LP or ULP state and perform continuous current and voltage measurements, the following tasks must be done:

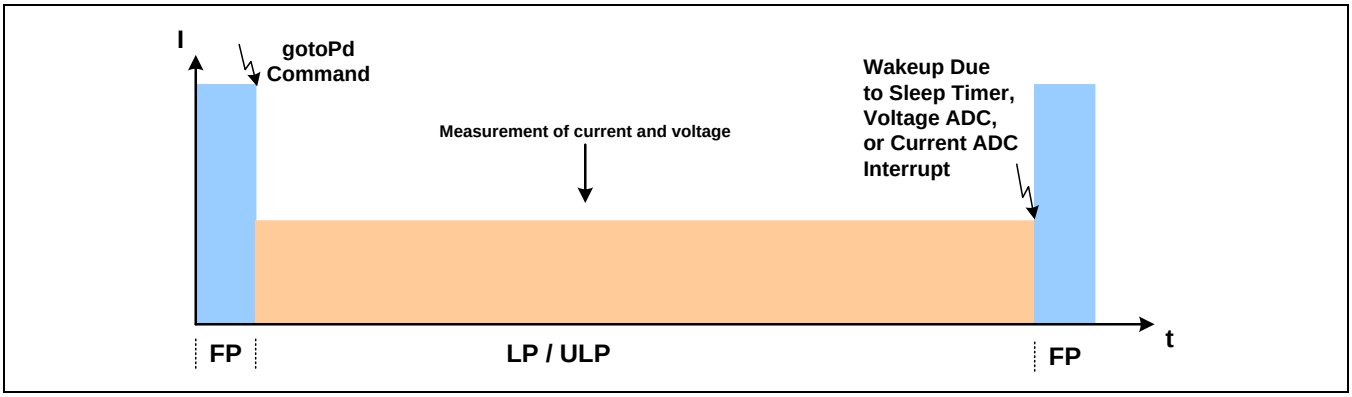
- Enable at least one of the following interrupts:
  - Set `irqEna[0]` to 1 to enable the watchdog interrupt to wake up the system.
  - Set `irqEna[1]` to 1 to enable the sleep timer to wake up the system.
  - Set `irqEna[4]` to 1 to enable the LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
  - Enable any ADC interrupt related to current.
- Set up the sleep timer compare value (register `sleepTCmp`) if needed.
- Set `pdState` to 0 or 1 to configure the LP state or to 2 to configure the ULP state.
- Set `pdMeas` to 5 to configure the system to perform continuous current and voltage measurements.
- Set `lpEnaLp`, `ulpEnaLp` and `pwrRefbufOcLp` as needed.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When an enabled interrupt occurs, the system wakes up and the settings from register `pwrCfgFp` are restored. When all blocks have stabilized, the MCU clock is re-enabled and if coming out of ULP state, the MCU reset is released.

Important: If any measurement is active while an enabled interrupt occurs (e.g., the sleep timer expires), the measurement is interrupted and the system returns to the FP state.

Figure 3.11 Performing Continuous Current and Voltage Measurements during LP/ULP State

Note: The sleep timer interrupt or an ADC interrupt related to voltage or current is used as the wake-up source in this example, but it could also be the watchdog timer interrupt or the LIN wakeup interrupt.



3.7.2.7. Performing Continuous Measurements of Current and Internal Temperature during LP/ULP State

This setup is the same as the configuration described in the previous section, except that the internal temperature instead of the voltage is measured. To use this option, `pdMeas` must be set to 6. Possible wake-up sources during this scenario are the watchdog timer interrupt, the sleep timer interrupt, the LIN wakeup interrupt, or any of the ADC interrupts related to current or temperature.

3.7.2.8. Performing Continuous Measurements of Current and External Temperature during LP/ULP State

This setup is the same as the configuration described in the previous section, except that the external temperature instead of the internal temperature is measured. To use this option, `pdMeas` must be set to 7. Possible wake-up sources during this scenario are the watchdog timer interrupt, the sleep timer interrupt, the LIN wakeup interrupt, or any of the ADC interrupts related to current or temperature.

3.7.3. OFF State

The OFF state is the power-down state with the lowest current consumption and no ADC measurements are possible. It is intended for long periods of inactivity; e.g., when a vehicle is shipped around the world. During this state, all oscillators and clocks are turned off, the MCU is not powered, and most of the analog blocks are powered down. Only the digital core and the RX part of the LIN PHY remain powered. The system can only wake up via the detection of a LIN wakeup frame. To go to the OFF state, the following tasks must be done:

- Set `irqEna[4]` to 1 to enable the LIN wake-up detector and to enable the system to wake up due to a LIN wakeup frame.
- Set `pdState` to 3 to configure the OFF state as the power-down state to be entered.
- Write $A9_{HEX}$ to register `gotoPd` and then drive the CSN line high.

When the LIN RXD line goes low during the OFF state, the low-power oscillator is re-enabled and the digital logic checks if the LIN RXD line is low for more than 150µs. If this is true, the complete system returns to FP state and the MCU is powered up, reset, and clocked again. If the LIN RXD line was low for less than 150µs, the low-power oscillator is powered down again and the system remains in the OFF state.

Important: If the LIN wakeup interrupt is not enabled, the system only can only wake up by a power-on reset.

3.7.4. Registers for Power Configuration and the Discreet Current Measurement Count

3.7.4.1. Register “pwrCfgFp” – Power Configuration Register for the FP State

Table 3.18 Register `pwrCfgFp`

Name	Address	Bits	Default	Access	Description
pwrAdcI	53_{HEX}	[0]	0_{BIN}	RW	When set to 1, the ADC for current is powered.
pwrAdcV		[1]	0_{BIN}	RW	When set to 1, the voltage/temperature ADC is powered.
Reserved		[2]	0_{BIN}	RW	Reserved; always write as 0.
lpEnaFp		[3]	0_{BIN}	RW	When set to 1, the bias current of the analog blocks is reduced to 10% in the FP state. **Note:** if `ulpEnaFp` is also set to 1, the bias current of the analog blocks is reduced to 15%.
ulpEnaFp		[4]	0_{BIN}	RW	When set to 1, the bias current of the analog blocks is reduced to 5% in the FP state. **Note:** if `lpEnaFp` is also set to 1, the bias current of the analog blocks is reduced to 15%.
pdRefbufOcFp		[5]	0_{BIN}	RW	When set to 1, the offset cancellation of the reference buffer is powered down.
Unused		[7:6]	00_{BIN}	RO	Unused; always write as 0.

3.7.4.2. Register “pwrCfgLp” – Power Configuration Register for Power-Down States

Table 3.19 Register `pwrCfgLp`

Name	Address	Bits	Default	Access	Description
pdState	64$_{HEX}$	[1:0]	00$_{BIN}$	RW	Select the power-down state to be entered: 0 or 1 – LP state 2 – ULP state 3 – OFF state
pdMeas		[4:2]	000$_{BIN}$	RW	Type of measurements to be performed during the LP or ULP state: 0 – No measurements 1 – Discrete measurements of current 2 – Discrete measurements of current, voltage, and internal temperature 3 – Discrete measurements of current, voltage, and external temperature 4 – Continuous measurements of current 5 – Continuous measurements of current and voltage 6 – Continuous measurements of current and internal temperature 7 – Continuous measurements of current and external temperature
lpEnaLp		[5]	1$_{BIN}$	RW	When set to 1, the bias current of the analog blocks is reduced to 10% in the LP/ULP state. **Note:** if `ulpEnaLp` is also set to 1, the bias current of the analog blocks is reduced to 15%.
ulpEnaLp		[6]	0$_{BIN}$	RW	When set to 1, the bias current of the analog blocks is reduced to 5% in the LP/ULP state. **Note:** if `lpEnaLp` is also set to 1, the bias current of the analog blocks is reduced to 15%.
pwrRefbufOcLp		[7]	0$_{BIN}$	RW	When set to 1, the offset cancellation of the reference buffer is powered in LP/ULP state while performing measurements.

3.7.4.3. Register “gotoPd” – Enter Power-Down State

Table 3.20 Register `gotoPd`

Name	Address	Bits	Default	Access	Description
gotoPd	65_{HEX}	[7:0]	00_{HEX}	WO	Writing $A9_{HEX}$ to this register triggers the PMU to enter the configured power-down state when the CSN line is driven high.

3.7.4.4. Register “discCvtCnt” – Configuration Register for Discrete Measurements

Table 3.21 Register `discCvtCnt`

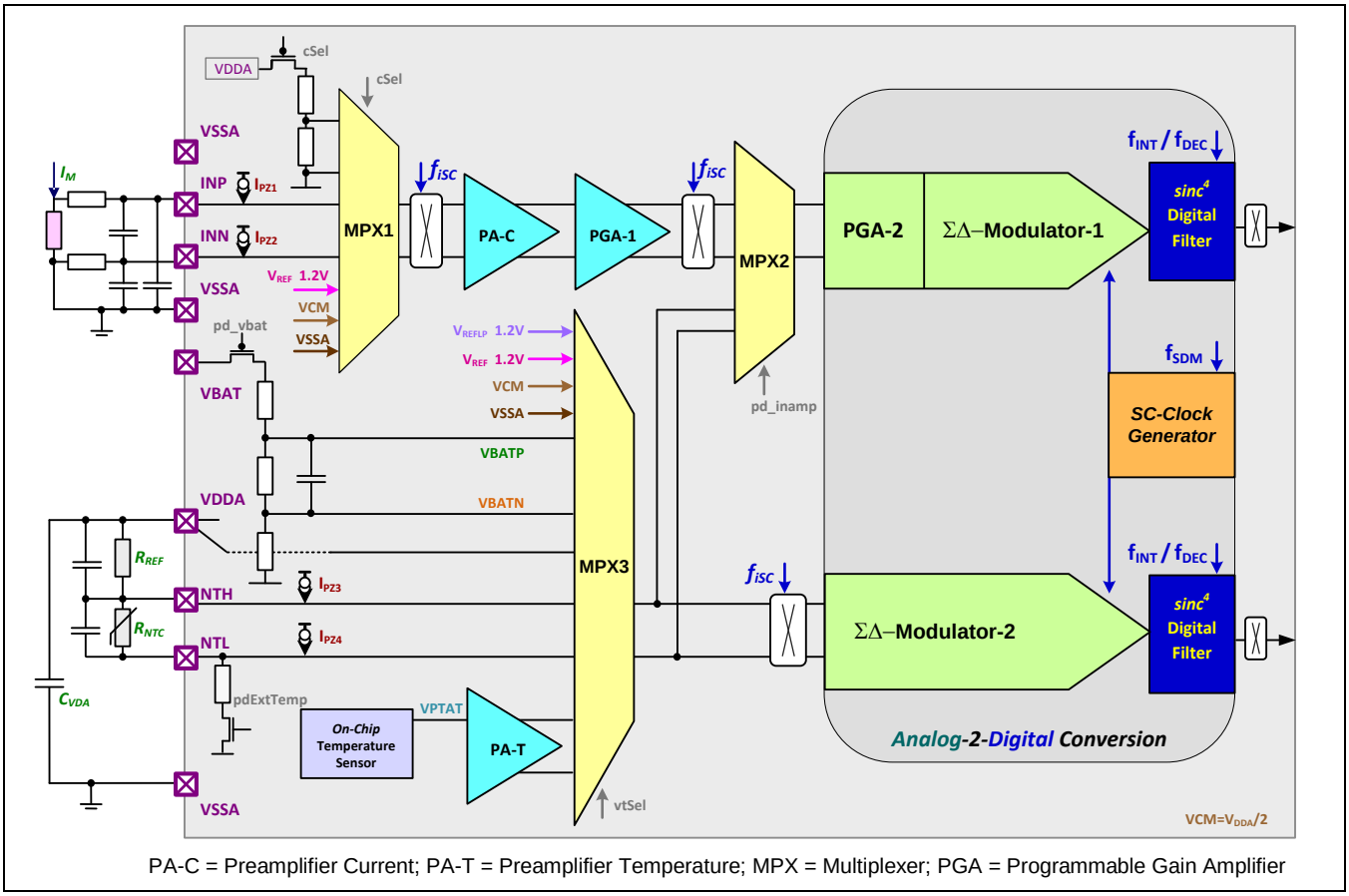
Name	Address	Bits	Default	Access	Description
discCvtCnt	$5F_{HEX}$	[7:0]	00_{HEX}	RW	Defines the number of "current only" measurements before performing one measurement of current, voltage, and temperature when `pdMeas` is 2 or 3.

3.8. SBC ADC Unit

The measurement subsystem incorporates two independent and synchronized high-resolution ADCs for monitoring two channels. The conversion scheme is based on the sigma-delta modulation (SDM) principle. One channel (ADC-I) is exclusively used for current measurement and includes a pre-amplifier with offset cancellation circuitry. The second channel (ADC-V/T) can be programmed to measure either voltage or temperature (internal or external).

The raw conversion data can be post-processed by calibration data to achieve a minimum offset and gain error (gain and offset correction). The conversion results are stored in the register file from where they can be read via the SPI digital communication interface. A completed conversion is flagged by a “data ready” signal that can be used as an interrupt source for the MCU. A detailed functional block diagram of the analog circuitry is shown in Figure 3.13.

Figure 3.12 Functional Block Diagram of the Analog Measurement Subsystem



PA-C = Preamplifier Current; PA-T = Preamplifier Temperature; MPX = Multiplexer; PGA = Programmable Gain Amplifier

The purpose of this analog architecture is to achieve a maximum level of diagnostic capability and flexibility as well as best accuracy.

The digital ADC unit consists of a data processing unit and control logic. The control logic generates the clocks and control signals for the analog SD-ADCs as well as control signals for the data processing part of the ADC unit.

3.8.1. ADC Clocks

Two clocks are generated in the digital part of the ADC unit and are driven to the analog part. The SMD clock is used for both SD-ADCs. The chop clock is used for the chopping operation within the SD-ADCs. The base for both clocks is the multiplexed clock muxClk, which is a 4MHz clock in FP state and a 125kHz clock in LP/ULP state.

3.8.1.1. ADC Clocks in FP State

In the FP state, the SDM clock is generated from the 4MHz clock by dividing it by two times the value programmed into the field `sdmClkDivFp` in register `sdmClkCfgFp` (see Table 3.24):

$$f_{SDM} = \frac{f_{HP}}{2 * sdmClkDivFp} \qquad f_{HP} = 4MHz \tag{4}$$

Important: When `sdmClkDivFp` is set to 0, the frequency of SDM clock is 2MHz.

The chop clock is generated from the SDM clock by further dividing it by 2, 4, 8, or 16 depending on the setting of the `sdmChopClkDiv` field in register `adcGomd` (see Table 3.55):

$$f_{CHOP} = f_{SDM} * 2^{-(sdmClkChopDiv+1)} \tag{5}$$

Although the clock bases used to generate the SDM clock and the chop clock have a frequency of 4MHz, the position of the clock edges used for the clock generation can be shifted relative to the 4MHz clock used for the digital logic to obtain optimal noise behavior for the analog part. The 4MHz clock used to generate the SDM clock ($CLK_{SDMBASE}$; see Figure 3.13 through Figure 3.16) is delayed relative to the 4MHz clock used for the digital logic (CLK_{MUXCLK}) by one to four 20MHz clock cycles (CLK_{HPOSC}) depending on the settings of the field `sdmPos` in register `sdmClkCfgFp` (see Table 3.24). The 4MHz clock used to generate the chop clock ($CLK_{CHOPBASE}$) is delayed relative to the 4MHz clock used for the digital logic ($CLK_{SDMBASE}$) by zero to four 20MHz clock cycles depending on the settings of field `sdmPos2` and field `sdmPos` in register `sdmClkCfgFp`. The delay in the number of 20MHz clock cycles of the chop clock to the SDM clock can be calculated using the following formula:

$$delay = (sdmPos2 - sdmPos) \bmod 5 \tag{6}$$

Important: The delay programmed into field `sdmPos2` is related to CLK_{MUXCLK}, not to $CLK_{SDMBASE}$. Table 3.22 shows the value that must be programmed into field `sdmPos2` depending on the field `sdmPos` and the desired delay.

Table 3.22 Value for sdmPos2 Depending on sdmPos and Desired Clock Delay from SDM to Chop Clocks

		sdmPos			
		0	1	2	3
Delay	0	0	1	2	3
	1	1	2	3	4
	2	2	3	4	0
	3	3	4	0	1
	4	4	0	1	2

Figure 3.13 FP ADC Clocking Scheme for `sdmPos` = `sdmPos2` = 2; `sdmClkDivFp` = 1; `sdmChopClkDiv`=0

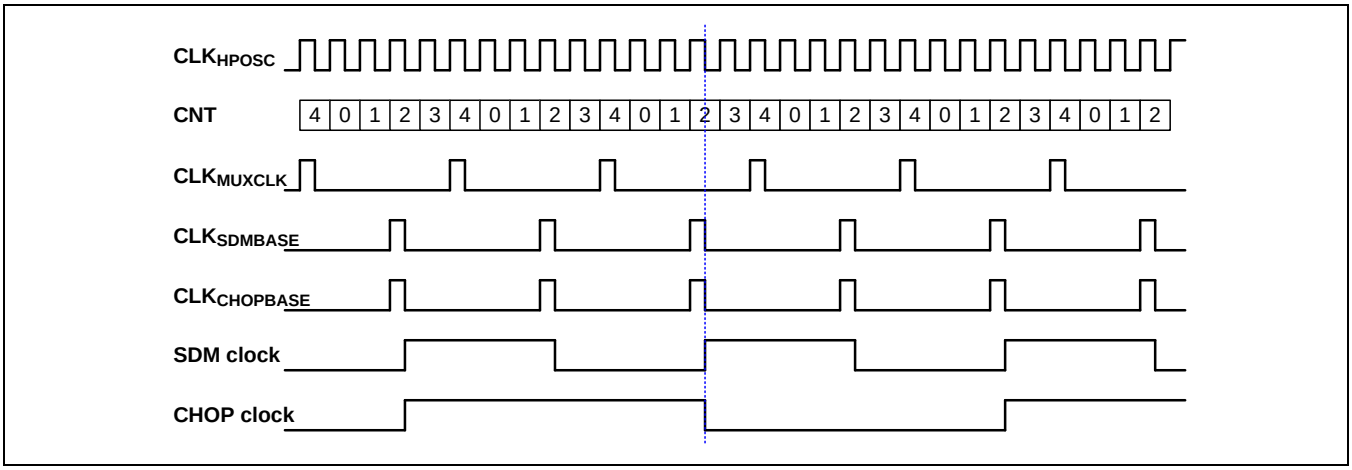


Figure 3.14 FP ADC Clocking for `sdmPos` = 1 and `sdmPos2` = 4; `sdmClkDivFp` = 1; `sdmChopClkDiv`=0

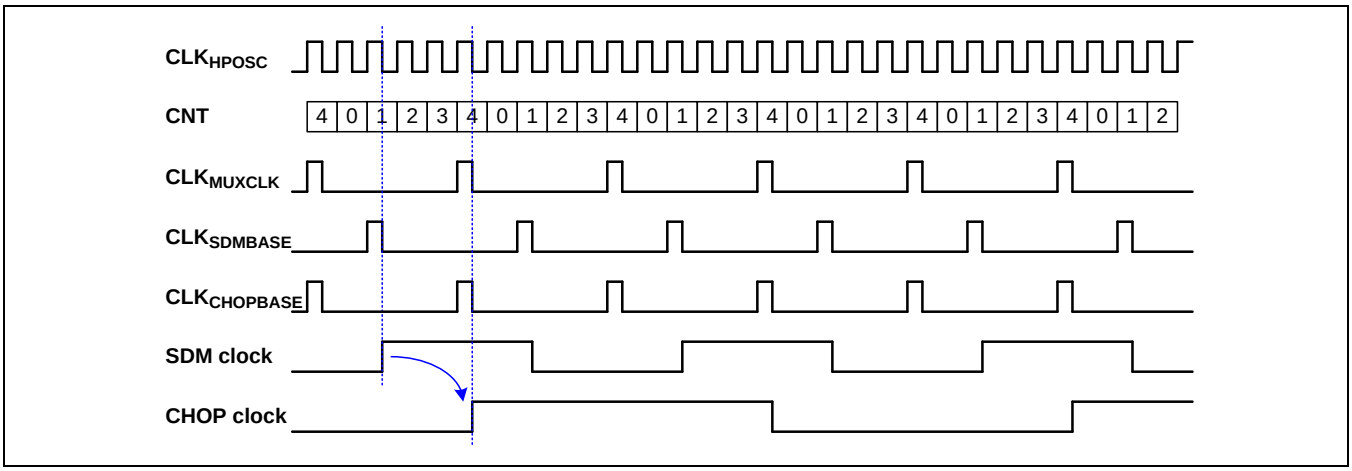


Figure 3.15 FP ADC Clocking for sdmPos = 3 and sdmPos2 = 0; sdmClkDivFp = 1; sdmChopClkDiv = 0

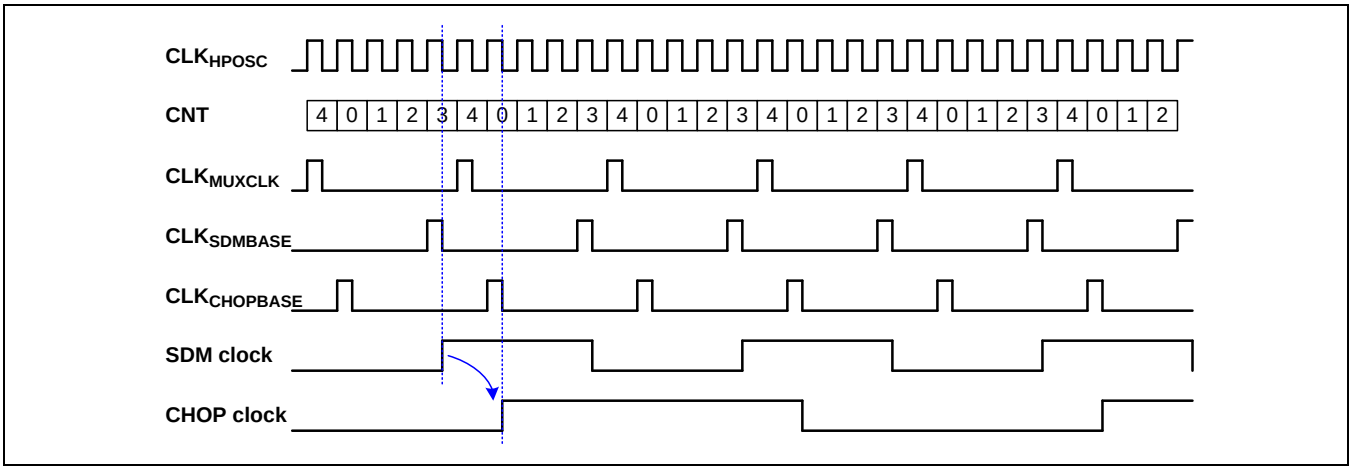
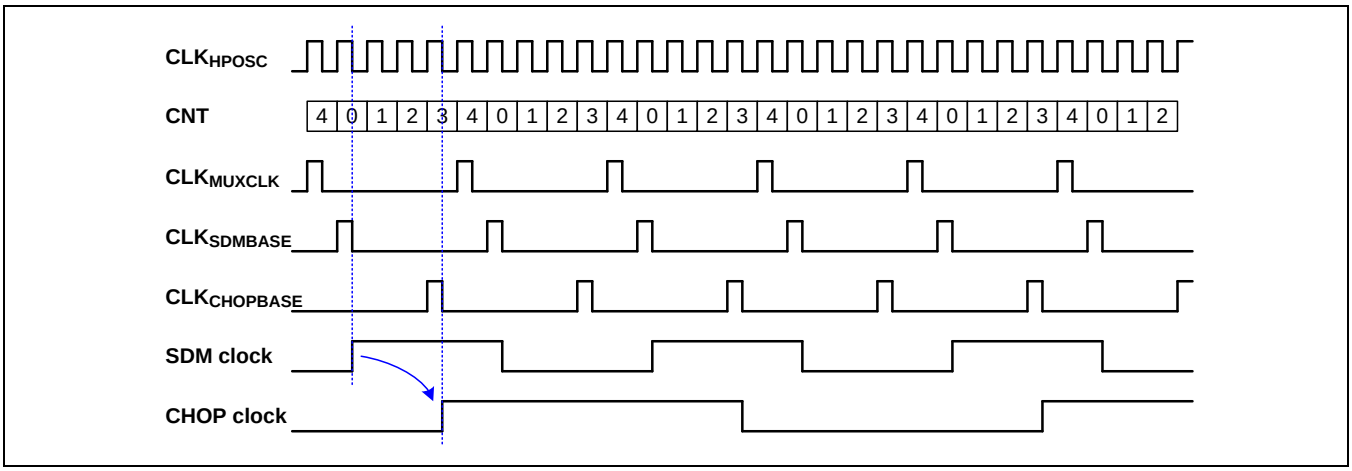


Figure 3.16 FP ADC Clocking for sdmPos = 0 and sdmPos2 = 3; sdmClkDivFp = 1; sdmChopClkDiv = 0



3.8.1.2. ADC Clocks in the LP/ULP State

In the LP or ULP state, the SDM clock is generated from the 125kHz clock (CLK_{LPOSC}) by dividing it by two times the value programmed into register field `sdmClkDivLp` (see Table 3.23):

$$f_{SDM} = \frac{f_{LP}}{2 * sdmClkDivLp}; \quad f_{LP} = 125\,kHz \tag{7}$$

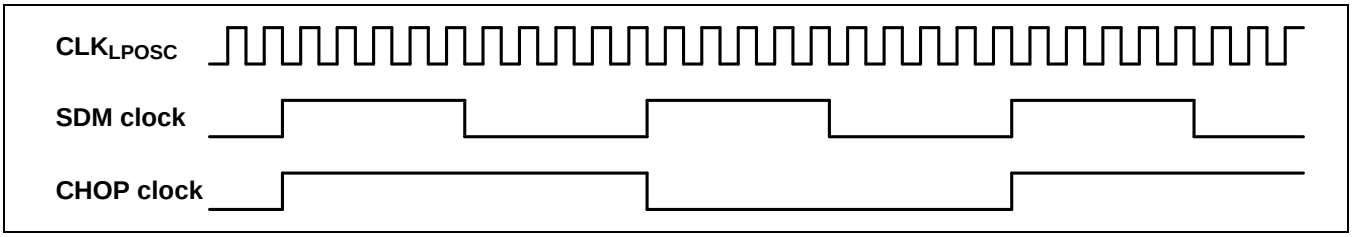
Important: When `sdmClkDivLp` is set to 0, the frequency of SDM clock is 62.5kHz.

The chop clock is generated from the SDM clock by further dividing it by 2, 4, 8, or 16 depending on the setting of the field `sdmChopClkDiv` in register `adcGomd` (see Table 3.55):

$$f_{CHOP} = f_{SDM} * 2^{-(sdmChopClkDiv+1)} \quad (8)$$

Both the SMD and chop clocks are generated from the same 125kHz clock that is used for the digital logic. Shifting of the clocks used to generate the SDM and chop clock is not possible and not needed as the analog clocks are generated on the falling clock edge where the digital logic is already stable and will not influence the analog part.

Figure 3.17 LP/ULP ADC Clocking Scheme; `sdmClkDivFp` = 5; `sdmChopClkDiv` = 0



3.8.1.3. Register "sdmClkCfgLp" – Configuration Register for the SDM Clocks in the LP/ULP State

Table 3.23 Register `sdmClkCfgLp`

Name	Address	Bits	Default	Access	Description
sdmClkDivLp[7:0]	$B0_{HEX}$	[7:0]	18_{HEX}	RW	Clock divider value for the SDM clock in the LP and ULP states related to the 125kHz base clock. With `sdmClkDivLp` = 0, the divider value is 2.
sdmClkDivLp[9:8]	$B1_{HEX}$	[1:0]	00_{BIN}	RW	
Unused		[7:2]	$00\ 0000_{BIN}$	RO	Unused; always write as 0.

3.8.1.4. Register "sdmClkCfgFp" – Configuration Register for the SDM Clocks in the FP State

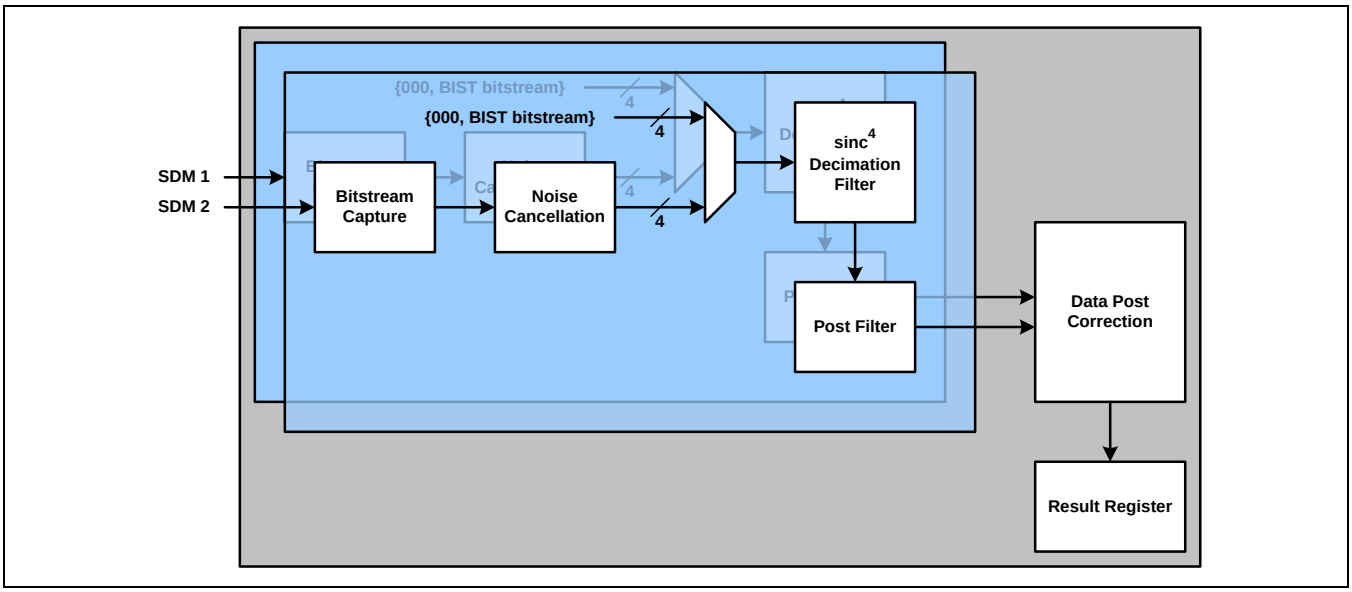
Table 3.24 Register `sdmClkCfgFp`

Name	Address	Bits	Default	Access	Description
sdmClkDivFp[7:0]	$B2_{HEX}$	[7:0]	08_{HEX}	RW	Clock divider value for the SDM clock in the FP state; related to the base clock. If 0, then the SDM clock is 2MHz.
sdmClkDivFp[9:8]	$B3_{HEX}$	[1:0]	00_{BIN}	RW	
Unused		[2]	0_{BIN}	RO	Unused; always write as 0.
sdmPos2		[5:3]	010_{BIN}	RW	Position of the chop clock relative to the CLK_{MUXCLK} clock.
sdmPos		[7:6]	10_{BIN}	RW	Position of the SDM clock relative to the base clock.

3.8.2. ADC Data Path

The incoming 2nd and 3rd order bit streams from the analog part of the SD-ADCs are first captured and then driven through a 3rd order noise shaping filter as illustrated in Figure 3.18. The digital conversion is accomplished by a 4th-order low-pass filter (sinc4 decimation filter). The bit stream capturing and the noise shaping filter cannot be directly changed by the user (no configuration registers), but the selected oversampling rate (register field `osr`) affects the sinc4 decimation filter (one output value per N input values).

Figure 3.18 Functional Block Diagram of the Digital ADC Data Path



A simple post filter (moving average filter) is placed behind the sinc4 decimation filter. The user can select the averaging function (no averaging, 2-stage averaging, or 3-stage averaging) via the `avgFiltCfg` bit field in the `adcSamp` register (see Table 3.56) when chopping is disabled. When chopping is enabled, the 2-stage averaging is used independently of the filter configuration.

The function of the 2-stage averaging filter is

$$x_{out}(t) = \frac{x_{in}(t) + x_{in}(t-1)}{2} \quad (9)$$

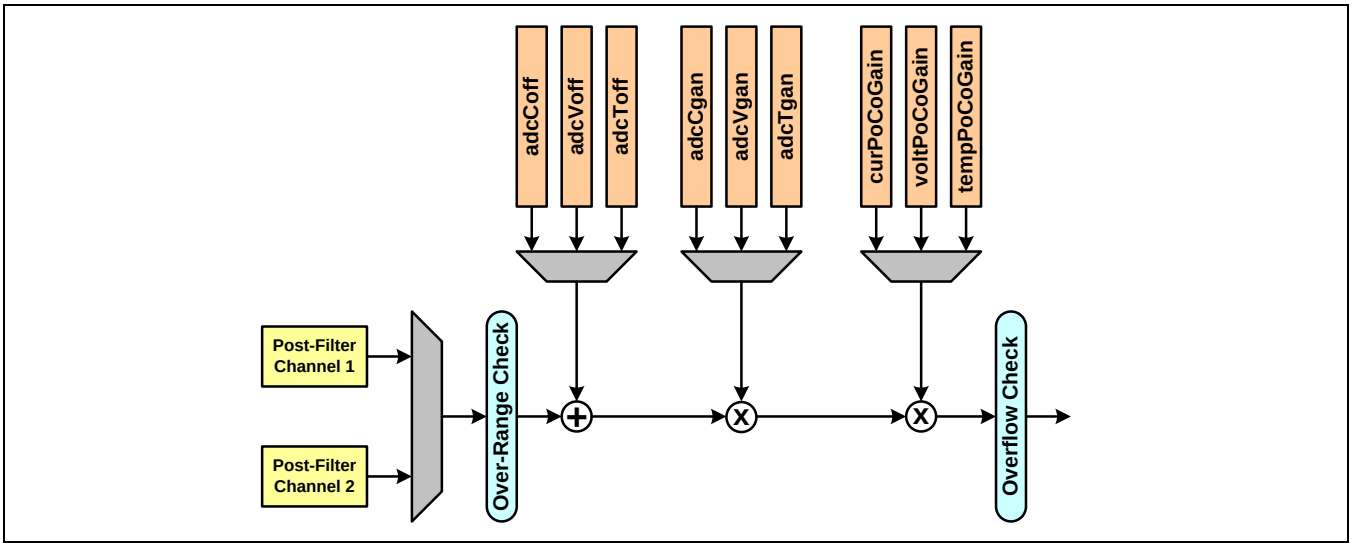
The function of the 3-stage averaging filter is

$$x_{out}(t) = \frac{x_{in}(t) + 2 * x_{in}(t-1) + x_{in}(t-2)}{4} \quad (10)$$

3.8.2.1. Data Post-Correction Block

The data post-correction block performs the offset and gain correction of the post-filtered conversion data as well as the over-range and the overflow detection.

Figure 3.19 Data Post Correction



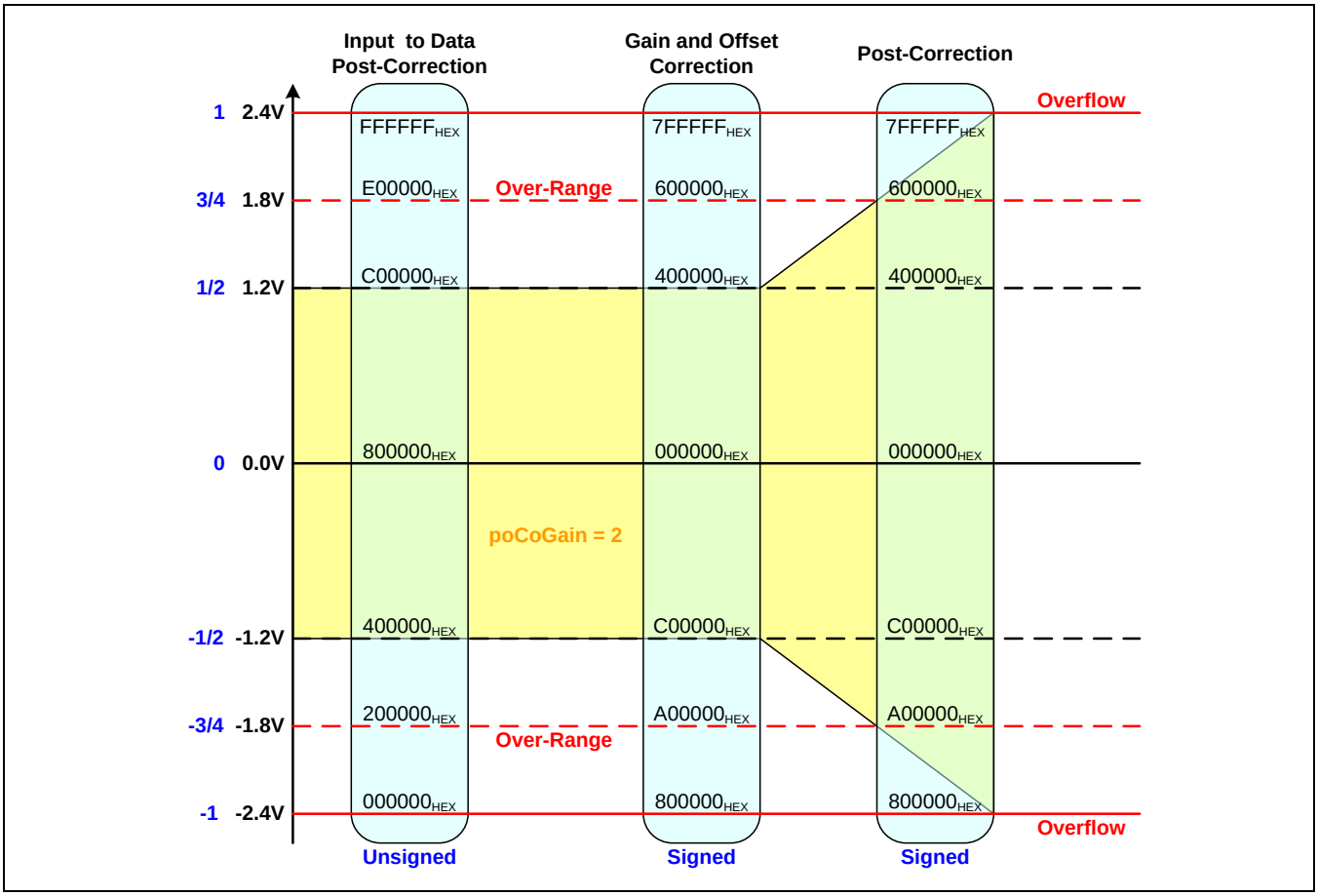
First, an over-range check is performed on the incoming data. Values that are outside the interval [-0.75; 0.75) are always mapped to the corresponding interval boundary. This is done for better results as the ADC accuracy decreases for large input values. The user can enable a "set interrupt" strobe for each of the two channels by setting the `adcAcmp` register bits `COvrEna` and `VTOvrEna` to 1 (see Table 3.54).

Note: The "set interrupt" strobes go to the interrupt controller. They have a different meaning than the corresponding "interrupt enable" bits (interrupt bits [15:14]) in the `irqEna` register. The "set interrupt" bits are used to select whether the interrupt status bits will be set or not; the "interrupt enable" bits select whether the interrupt status bits will drive the interrupt line or not.

After the over-range check, a programmable offset, interpreted as a number in range [-1.0; 1.0), is added to the data. Three registers allow setting different offsets for current, voltage, and temperature: `adcCoff`, `adcVoff`, and `adcToff` (see Table 3.25, Table 3.27, and Table 3.29 respectively). The offset correction is followed by two multiplication stages. In the first multiplication stage, individual gain factors for current (`adcCgan`), voltage (`adcVgan`), or temperature (`adcTgan`), interpreted as numbers in the range [0.0; 2.0), are multiplied by the offset corrected data (see Table 3.26, Table 3.28, and Table 3.30 respectively). The second multiplication stage is used to shift the significant data into the most significant bits of the result register. The data is multiplied by 1, 2, 4, or 8, which can be individually selected for current, voltage, and temperature via the `curPoCoGain`, `voltPoCoGain`, and `tempPoCoGain` fields in the `adcPoCoGain` register (see Table 3.31).

Figure 3.20 Data Representation through Data Post Correction including Over-Range and Overflow Levels

Note: the yellow area represents the usable data space to avoid overflow when the post correction gain is 2.

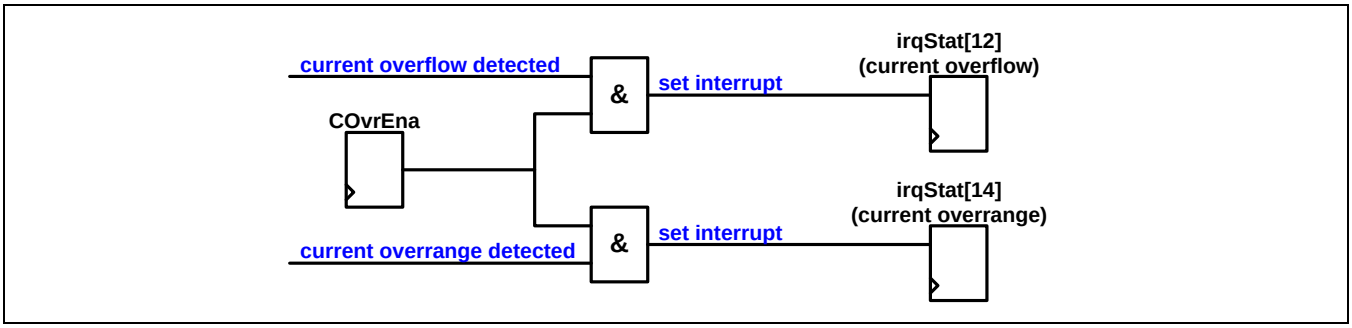


An overflow check is performed on the output of the second multiplication stage as the result could be out of the representable range of [-1.0; 1.0). The user can also enable a “set interrupt” strobe for each of the two channels by setting the `adcAcmp` register bits `CovrEna` and `VTOvrEna` to 1 (same bits as for the over-range check).

Note: Although the same “set interrupt strobe enable” bits are used for over-range and overflow, independent interrupt status bits exist that can be individually enabled or disabled for overflow (interrupt bits [13:12]).

Figure 3.21 illustrates the common enable `CovrEna` for the interrupt strobes for current over-range and overflow. The function of `VTOvrEna` as the common enable for the interrupt strobes for voltage/temperature over-range and overflow conditions is similar.

Figure 3.21 Common Enable for the "set overrange" and "set overflow" Interrupt Strobes for Current



3.8.2.2. Register "adcCoff" – Offset Correction Value for Current Channel

Table 3.25 Register adcCoff

Name	Address	Bits	Default	Access	Description
adcCoff[7:0]	33_{HEX}	[7:0]	00_{HEX}	RW	Offset value for current value in 2's complement representation; interpreted as a number in the range of [-1.0; 1.0), programmable offset range = +/- 2 V_{REF}, V_{REF}= full-scale range ADC. **Note:** The initial value is loaded from OTP after reset.
adcCoff[15:8]	34_{HEX}	[7:0]	00_{HEX}	RW	
adcCoff[23:16]	35_{HEX}	[7:0]	00_{HEX}	RW	

3.8.2.3. Register "adcCgan" – Gain Correction Value for Current Channel

Table 3.26 Register adcCgan

Name	Address	Bits	Default	Access	Description
adcCgan[7:0]	30_{HEX}	[7:0]	00_{HEX}	RW	Gain value for current value; interpreted as a number in the range [0.0; 2.0). **Note:** The initial value is loaded from OTP after reset.
adcCgan[15:8]	31_{HEX}	[7:0]	00_{HEX}	RW	
adcCgan[23:16]	32_{HEX}	[7:0]	80_{HEX}	RW	

3.8.2.4. Register "adcVoff" – Offset Correction Value for Voltage Channel

Table 3.27 Register adcVoff

Name	Address	Bits	Default	Access	Description
adcVoff[7:0]	39_{HEX}	[7:0]	00_{HEX}	RW	Offset value for voltage value in 2's complement representation; interpreted as a number in the range of [-1.0; 1.0), programmable offset range = +/- 2 V_{REF}, V_{REF} = full-scale range ADC. **Note:** The initial value is loaded from OTP after reset.
adcVoff[15:8]	$3A_{HEX}$	[7:0]	00_{HEX}	RW	
adcVoff[23:16]	$3B_{HEX}$	[7:0]	00_{HEX}	RW	

3.8.2.5. Register “adcVgan” – Gain Correction Value for Voltage Channel

Table 3.28 Register adcVgan

Name	Address	Bits	Default	Access	Description
adcVgan[7:0]	36_{HEX}	[7:0]	00_{HEX}	RW	Gain value for voltage value; interpreted as a number in the range [0.0; 2.0). **Note:** The initial value is loaded from OTP after reset.
adcVgan[15:8]	37_{HEX}	[7:0]	00_{HEX}	RW	
adcVgan[23:16]	38_{HEX}	[7:0]	80_{HEX}	RW	

3.8.2.6. Register “adcToff” – Offset Correction Value for Temperature Channel

Table 3.29 Register adcToff

Name	Address	Bits	Default	Access	Description
adcToff[7:0]	$3E_{HEX}$	[7:0]	00_{HEX}	RW	Offset value for temperature value; interpreted as a number in the range [-1.0; 1.0) **Note:** The initial value is loaded from OTP after reset.
adcToff[15:8]	$3F_{HEX}$	[7:0]	00_{HEX}	RW	

3.8.2.7. Register “adcTgan” – Gain Correction Value for Temperature Channel

Table 3.30 Register adcTgan

Name	Address	Bits	Default	Access	Description
adcTgan[7:0]	$3C_{HEX}$	[7:0]	00_{HEX}	RW	Gain value for temperature value; interpreted as a number in the range [0.0; 2.0) **Note:** The initial value is loaded from OTP after reset.
adcTgan[15:8]	$3D_{HEX}$	[7:0]	80_{HEX}	RW	

3.8.2.8. Register “adcPoCoGain” – Post Correction Gain Configuration

Table 3.31 Register adcPoCoGain

Name	Address	Bits	Default	Access	Description
curPoCoGain	57_{HEX}	[1:0]	00_{BIN}	RW	Post correction gain for the current channel: 0 – Gain factor is 1 1 – Gain factor is 2 2 – Gain factor is 4 3 – Gain factor is 8
voltPoCoGain		[3:2]	00_{BIN}	RW	Post correction gain for the voltage channel: 0 – Gain factor is 1 1 – Gain factor is 2 2 – Gain factor is 4 3 – Gain factor is 8
tempPoCoGain		[5:4]	00_{BIN}	RW	Post correction gain for the temperature channel: 0 – Gain factor is 1 1 – Gain factor is 2 2 – Gain factor is 4

Name	Address	Bits	Default	Access	Description
					3 Gain factor is 8
Unused		[7:6]	00_{BIN}	RO	Unused; always write as 0.

3.8.3. ADC Operating Modes and Related Registers

3.8.3.1. Single Measurement Results

Each value coming from the data post correction block is the result of a single measurement. These values are signed and stored in the corresponding result registers `adcCdat`, `adcVdat`, `adcTdat`, or `adcRdat`. The following formulas can be used to calculate the battery current and the battery voltage from the result register values:

$$I_{BAT} = \frac{adcCdat * 2 * V_{REF}}{R_{SHUNT} * 2^{23} * G_{ANA} * G_{POCO}} \quad (11)$$

$$V_{BAT} = \frac{adcVdat * 24 * 2 * V_{REF}}{2^{23} * G_{POCO}} \quad (12)$$

Where

I_{BAT}	Battery current
V_{BAT}	Battery voltage
G_{ANA}	Analog gain in current path (pgaIfc ∗ pga1 ∗ pga2; see Table 3.36)
G_{POCO}	Digital gain in post-correction stage (second multiplication; see Table 3.31)
R_{SHUNT}	Shunt resistance
V_{REF}	Reference voltage
adcCdat	Register value for current
adcVdat	Register value for voltage

3.8.3.2. Register “adcCdat” – Single Current Measurement Value

Table 3.32 Register adcCdat

Name	Address	Bits	Default	Access	Description
adcCdat[7:0]	02_{HEX}	[7:0]	00_{HEX}	RO	Conversion result of a single current measurement (signed value)
adcCdat[15:8]	03_{HEX}	[7:0]	00_{HEX}	RO	
adcCdat[23:16]	04_{HEX}	[7:0]	00_{HEX}	RO	

3.8.3.3. Register “adcVdat” – Single Voltage Measurement Value

Table 3.33 Register adcVdat

Name	Address	Bits	Default	Access	Description
adcVdat[7:0]	05_{HEX}	[7:0]	00_{HEX}	RO	Conversion result of a single voltage measurement (signed value)
adcVdat[15:8]	06_{HEX}	[7:0]	00_{HEX}	RO	
adcVdat[23:16]	07_{HEX}	[7:0]	00_{HEX}	RO	

3.8.3.4. Registers “adcTdat” and “adcRdat” – Single Temperature Measurement Values

Table 3.34 Register adcTdat

Name	Address	Bits	Default	Access	Description
adcTdat[7:0]	$0A_{HEX}$	[7:0]	00_{HEX}	RO	Conversion result of a single temperature value (signed value; inverted); this value is either the internally measured temperature or the NTC value of an external temperature measurement. **Important:** This value is sign-inverted.
adcTdat[15:8]	$0B_{HEX}$	[7:0]	00_{HEX}	RO	

Table 3.35 Register adcRdat

Name	Address	Bits	Default	Access	Description
adcRdat[7:0]	08_{HEX}	[7:0]	00_{HEX}	RO	ADC result register of a single temperature measurement by reading a voltage across the reference resistor (external temperature measurement only).
adcRdat[15:8]	09_{HEX}	[7:0]	00_{HEX}	RO	

3.8.3.5. Register “adcGain” – Analog Gain Configuration in the Current Path

Table 3.36 Register adcGain

Name	Address	Bits	Default	Access	Description
pgaIfc	52_{HEX}	[1:0]	00_{BIN}	RW	Sets the gain of the IFC in the analog current path: 0 – Gain factor is 1 1 – Gain factor is 2 2 – Gain factor is 4 3 – Gain factor is 8
pga1		[3:2]	00_{BIN}	RW	Sets the gain of the PGA1 in the analog current path: 0 – Gain factor is 1 1 – Gain factor is 2 2 – Gain factor is 4 3 – Gain factor is 8
pga2		[4]	0_{BIN}	RW	Sets the gain of the PGA2 in the analog current path: 0 – Gain factor is 4 1 – Gain factor is 8
Unused		[7:5]	000_{BIN}	RO	Unused; always write as 0.

3.8.3.6. Result Counter Functionality and Conversion Ready Strobes

Three status bits are available in the interrupt status (`irqStat[7:5]`) that signal that the conversion of current, voltage, or temperature has completed. The "set interrupt" strobe is generated for each completed temperature measurement. For the voltage and current measurements, the user can independently select whether the corresponding "set interrupt" strobe will be generated after each single measurement (SRCS – single result count sequence) or after N measurements (MRCS – multi-result count sequence).

The register `adcCrcl` configures the number of current measurements before the current conversion ready strobe is generated; the maximum number is 65535. Setting this register to 0 disables the result count functionality which means that SRCS is configured. The present result counter value can always be read from the register `adcCrcv`. The result counter is reset when `startAdcC` is set (rising edge) in FP state (see Table 3.58) or at start of the first measurement in LP/ULP state. It is set to 1 at the end of the first measurement after the limit defined in `adcCrcl` has been reached.

The register `adcVrcl` configures the number of measurements before the voltage conversion ready strobe is generated, the maximum number is 15. Setting this register to 0 disables the result count functionality, which means that SRCS is configured. The present result counter value can always be read from the register `adcVrcv`. The result counter is reset when `startAdcV` is set (rising edge) in the FP state (see Table 3.58) or at the start of the first measurement in the LP/ULP state. It is set to 1 at the end of the first measurement after the limit defined in `adcVrcl` was reached.

Note: Setting register `adcCrcl` or `adcVrcl` to 1 leads to SRCS in the corresponding channel.

3.8.3.7. Register "adcCrcl" – Current Result Count Limit

Table 3.37 Register `adcCrcl`

Name	Address	Bits	Default	Access	Description
adcCrcl[7:0]	40_{HEX}	[7:0]	00_{HEX}	RW	Number of current measurements before the current conversion ready strobe is generated. **Note:** Setting this bit to 0 disables this functionality, and the strobe is generated after each current measurement.
adcCrcl[15:8]	41_{HEX}	[7:0]	00_{HEX}	RW	

3.8.3.8. Register "adcCrcv" – Current Result Count Value

Table 3.38 Register `adcCrcv`

Name	Address	Bits	Default	Access	Description
adcCrcv[7:0]	$1B_{HEX}$	[7:0]	00_{HEX}	RO	Present value of the current result counter.
adcCrcv[15:8]	$1C_{HEX}$	[7:0]	00_{HEX}	RO	

3.8.3.9. Register “adcVrcl” – Voltage Result Count Limit

Table 3.39 Register adcVrcl

Name	Address	Bits	Default	Access	Description
adcVrcl	45_{HEX}	[3:0]	0000_{BIN}	RW	Number of voltage measurements before the voltage conversion ready strobe is generated. **Note:** Setting this bit to 0 disables this functionality, and the strobe is generated after each voltage measurement.
Unused		[7:4]	0000_{BIN}	RO	Unused; always write as 0.

3.8.3.10. Register “adcVrcv” – Voltage Result Count Value

Table 3.40 Register adcVrcv

Name	Address	Bits	Default	Access	Description
adcVrcv	$1E_{HEX}$	[3:0]	0000_{BIN}	RO	Present value of the voltage result counter.
Unused		[7:4]	0000_{BIN}	RO	Unused; always write as 0.

3.8.3.11. Current Threshold Comparator Functionality

The current threshold comparator functionality is used to monitor the current level and to generate an interrupt (`irqStat[8]`) if the absolute current value exceeds a programmable limit for a configurable number of conversion results. This functionality is enabled when the field `ctcvMode` in register `adcAcmp` is set to a non-zero value. If enabled, this function is always triggered when a new current value is measured. The absolute value of the most significant 17 bits of the measured current value is compared to the expanded programmable threshold register `adcCrth` (see Table 3.41):

$$\mathrm{abs}(\mathrm{adcCdat}[23:7]) \geq \{0, \mathrm{adcCrth}\} \quad (13)$$

When the current threshold comparator functionality is enabled, the current threshold counter is used to count the number of conversions where the absolute current value is above the threshold. If the absolute current value is greater than or equal to the programmed threshold (above formula is true), the internal current threshold counter is incremented (until it reaches its maximum value FF_{HEX}). Otherwise the counter is either decremented (if `ctcvMode` field in register `adcAcmp` is set to 1) or reset (if `ctcvMode` is set to 2), or it remains unchanged (if `ctcvMode` is set to 3). The present value of the current threshold counter can be read from the register `adcCtcv` (see Table 3.43).

Note: When bit field `ctcvMode` is set to 00_{BIN}, the current threshold comparator functionality is disabled and register `adcCrtv` is always 0.

Note: When `ctcvMode` is set to 01_{BIN}, the current threshold counter is not decremented when the counter is 0.

After each comparison of the absolute current value versus the current threshold level and after the current threshold counter has been updated, the internal current threshold counter is compared to the current threshold counter limit (register `adcCtcl;` see Table 3.42). Whenever the current threshold counter is greater than or equal to the programmable limit, a “set interrupt” strobe is generated.

Note: When the current threshold counter has reached its limit and it is configured to keep its value if the limit is not reached, a "set interrupt" strobe is generated for each new measurement even if the new value is below threshold.

The current threshold counter is reset to 0 for the following conditions:

- If `ctcvMode` is set to 2 and the absolute current value is below the programmed threshold `adcCrth`
- On assertion of `startAdcI` (rising edge) in the FP state
- At the start of the first conversion in the LP or ULP state
- Each time the result counter is reset (if the result counter is enabled) and the current threshold counter reset mode bit (bit `ctcvRstMode` in register `adcAcmp`) is set to 1

3.8.3.12. Register "adcCrth" – Absolute Current Threshold

Table 3.41 Register `adcCrth`

Name	Address	Bits	Default	Access	Description
adcCrth[7:0]	42_{HEX}	[7:0]	00_{HEX}	RW	Absolute current threshold (unsigned value). When using current comparator threshold functionality, the absolute current value is compared to {0, adcCrth}.
adcCrth[15:8]	43_{HEX}	[7:0]	00_{HEX}	RW	

3.8.3.13. Register "adcCtcl" – Current Threshold Counter Limit

Table 3.42 Register `adcCtcl`

Name	Address	Bits	Default	Access	Description
adcCtcl	44_{HEX}	[7:0]	00_{HEX}	RW	Current threshold counter limit. This register defines the number of current measurements that must be greater than or equal to the threshold "adcCrth" before the interrupt is set.

3.8.3.14. Register "adcCtcv" – Current Threshold Counter Value

Table 3.43 Register `adcCtcv`

Name	Address	Bits	Default	Access	Description
adcCtcv	$1D_{HEX}$	[7:0]	00_{HEX}	RO	Present current threshold counter value.

3.8.3.15. Current Accumulator Functionality

The current accumulator functionality is used to sum up all current conversion results. The present accumulator value can be read from the register `adcCaccu` (signed value; see Table 3.45). Positive conversion results increment the accumulator register; negative conversion results decrement it. The accumulator register saturates at its minimum and maximum value.

The current accumulator is reset to 0 under these conditions:

- On assertion of `startAdcI` (rising edge) in the FP state
- At start of the first conversion in the LP or ULP state
- Each time the result counter is reset (if the result counter is enabled) and the current accumulator reset mode bit (bit `accuRstMode` in register `adcAcmp`) is set to 1

Note: The current accumulator functionality can be used to calculate the mean value of the current.

The current accumulator is also compared to a programmable signed accumulator threshold value (register `adcCaccTh`). This comparison can be used to generate a "set interrupt" strobe for `irqStat[11]`, but to enable the generation of the "set interrupt" strobe, bit `CaccuThEna` in register `adcAcmp` must be set to 1. The "set interrupt" strobe is always generated on update of the accumulator register when

- `adcCaccTh` is greater than 0 and `adcCaccu` is greater than `adcCaccTh`
- `adcCaccTh` is lower than 0 and `adcCaccu` is lower than `adcCaccTh`
- `adcCaccTh` is equal to 0 and `adcCaccu` is not equal to 0

3.8.3.16. Register "adcCaccTh" – Current Accumulator Threshold Value

Table 3.44 Register adcCaccTh

Name	Address	Bits	Default	Access	Description
adcCaccTh[7:0]	48_{HEX}	[7:0]	00_{HEX}	RW	Signed threshold value for current accumulator mode.
adcCaccTh[15:8]	49_{HEX}	[7:0]	00_{HEX}	RW	
adcCaccTh[23:16]	$4A_{HEX}$	[7:0]	00_{HEX}	RW	
adcCaccTh[31:24]	$4B_{HEX}$	[7:0]	00_{HEX}	RW	

3.8.3.17. Register "adcCaccu" – Current Accumulator Value

Table 3.45 Register adcCaccu

Name	Address	Bits	Default	Access	Description
adcCaccu[7:0]	$0C_{HEX}$	[7:0]	00_{HEX}	RO	Present current accumulator value.
adcCaccu[15:8]	$0D_{HEX}$	[7:0]	00_{HEX}	RO	
adcCaccu[23:16]	$0E_{HEX}$	[7:0]	00_{HEX}	RO	
adcCaccu[31:24]	$0F_{HEX}$	[7:0]	00_{HEX}	RO	

3.8.3.18. Voltage Threshold Comparator and Voltage Accumulator Functionality

The ZSSC1956 also provides a threshold comparator as well as an accumulator comparator for the battery voltage channel but with reduced functionality.

When the VthSel bit in register adcAcmp is set to 0, the absolute value of the most significant 17 bits of a single voltage measurement (register adcVdat) is compared to the programmable voltage threshold (register adcVTh). In this case, register adcVTh is interpreted as an unsigned value. There is also no counter functionality. Whenever the absolute voltage value is below the programmed threshold, a "set interrupt" strobe for irqStat[9] is generated when the strobe generation is enabled (field VthWuEna in register adcAcmp is set to 1).

$$\mathrm{abs}(\mathrm{adcVdat}[23:7]) < \{0, \mathrm{adcVTh}\} \tag{14}$$

When bit VthSel in register adcAcmp is set to 1, the voltage accumulator functionality is enabled. The voltage result counter functionality must also be enabled (register adcVrcl > 0). The voltage accumulator functionality is used to sum up all voltage conversion results. In contrast to the current channel, only the upper 20 bits of the voltage conversion results are accumulated. The present accumulator value can be read from the register adcVaccu (signed value; see Table 3.47). Positive conversion results increment the accumulator register; negative conversion results decrement it. The accumulator register saturates at its minimum and maximum value.

The voltage accumulator is reset to 0 under these conditions:

- On assertion of startAdcV (rising edge) in the FP state
- At start of the first conversion in the LP or ULP state
- Each time the result counter is reset (if the result counter is enabled)

Note: The voltage accumulator functionality can be used to calculate the mean value of the voltage.

After the last accumulation within an MRCS, the upper 16 bits of the voltage accumulator are compared to the voltage threshold adcVTh, which is interpreted as a signed value in this case. This comparison can be used to generate a "set interrupt" strobe for irqStat[9], but to enable the generation of the "set interrupt" strobe, bit VthWuEna in register adcAcmp must be set to 1. The "set interrupt" strobe is generated when

- adcVTh is greater than 0 and adcVaccu is less than or equal to adcVTh
- adcVTh is lower than 0 and adcVaccu is greater than or equal to adcVTh
- adcVTh is equal to 0 and adcVaccu is equal to 0

Important: The threshold adcVTh is either interpreted as an unsigned or signed value depending on the operation mode (VthSel).

Note: the voltage comparators compare only on the MSBs of the conversion result, so it might be beneficial to use the post correction gain functionality to shift left the results to increase the accuracy of the comparison.

3.8.3.19. Register “adcVTh” – Voltage Threshold Value

Table 3.46 Register adcVTh

Name	Address	Bits	Default	Access	Description
adcVTh[7:0]	46_{HEX}	[7:0]	00_{HEX}	RW	Voltage threshold. If `VthSel == 0`, then `adcVTh` is interpreted as an unsigned value and it is compared to the absolute value of a single voltage conversion. If `VthSel == 1`, then `adcVTh` is interpreted as a signed value and it is compared to the accumulated voltage conversion results at the end of an MRCS.
adcVTh[15:8]	47_{HEX}	[7:0]	00_{HEX}	RW	

3.8.3.20. Register “adcVaccu” – Voltage Accumulator Value

Table 3.47 Register adcVaccu

Name	Address	Bits	Default	Access	Description
adcVaccu[7:0]	10_{HEX}	[7:0]	00_{HEX}	RO	Present voltage accumulator value.
adcVaccu[15:8]	11_{HEX}	[7:0]	00_{HEX}	RO	
adcVaccu[23:16]	12_{HEX}	[7:0]	00_{HEX}	RO	

3.8.3.21. Minimum and Maximum Values of Current and Voltage

For current and voltage measurements, the minimum and maximum values are determined on the upper 16 bits of the corresponding conversion results. These values can be read from registers `adcCmax`, `adcCmin`, `adcVmax`, and `adcVmin`. These registers are reset in the same manner as the corresponding accumulator registers. These values are only provided for statistical reasons and can be used to judge the accumulated current or voltage values when used for mean value calculation.

Note: As the minimum and maximum values are only determined on the MSBs of the corresponding conversion results, it might be beneficial to use the post correction gain functionality to shift left the results to increase the accuracy of the comparison.

3.8.3.22. Register “adcCmax” – Maximum Current Value

Table 3.48 Register adcCmax

Name	Address	Bits	Default	Access	Description
adcCmax[7:0]	13_{HEX}	[7:0]	00_{HEX}	RO	Upper 16 bits of the maximum measured current value (signed value).
adcCmax[15:8]	14_{HEX}	[7:0]	80_{HEX}	RO	

3.8.3.23. Register “adcCmin” – Minimum Current Value

Table 3.49 Register adcCmin

Name	Address	Bits	Default	Access	Description
adcCmin[7:0]	15_{HEX}	[7:0]	FF_{HEX}	RO	Upper 16 bits of the minimum measured current value (signed value).
adcCmin[15:8]	16_{HEX}	[7:0]	$7F_{HEX}$	RO	

3.8.3.24. Register “adcVmax” – Maximum Voltage Value

Table 3.50 Register adcVmax

Name	Address	Bits	Default	Access	Description
adcVmax[7:0]	17_{HEX}	[7:0]	00_{HEX}	RO	Upper 16 bits of the maximum measured voltage value (signed value).
adcVmax[15:8]	18_{HEX}	[7:0]	80_{HEX}	RO	

3.8.3.25. Register “adcVmin” – Minimum Voltage Value

Table 3.51 Register adcVmin

Name	Address	Bits	Default	Access	Description
adcVmin[7:0]	19_{HEX}	[7:0]	FF_{HEX}	RO	Upper 16 bits of the minimum measured voltage value (signed value).
adcVmin[15:8]	$1A_{HEX}$	[7:0]	$7F_{HEX}$	RO	

© 2016 Integrated Device Technology, Inc.

3.8.3.26. Temperature Limits

The user can define an upper (register `adcTmax`) and a lower (register `adcTmin`) limit for the external and internal temperature measurement. On each update of register `adcTdat` (see Table 3.34), the upper 8 bits are compared to the signed limit values. This can be used to generate a "set interrupt" strobe for `irqStat[10]` if the value for `adcTdat` is outside the interval [`adcTmin`; `adcTmax`] and the `TwuEna` bit in register `adcAcmp` has been set to 1.

Note: The minimum and maximum values are only compared to the MSBs of the conversion result, so it might be beneficial to use the post correction gain functionality to shift left the results to increase the accuracy of the comparison.

Important: Because the value stored in register `adcTdat` is inverted, the value given in register `adcTmax` is actually the value for the lower temperature and the value given in register `adcTmin` is actually the value for the higher temperature.

3.8.3.27. Register "adcTmax" – Upper Boundary for Temperature Interval

Table 3.52 Register `adcTmax`

Name	Address	Bits	Default	Access	Description
adcTmax	$4C_{HEX}$	[7:0]	00_{HEX}	RW	Upper boundary for the temperature interval compared to the upper bits of `adcTdat`.

3.8.3.28. Register "adcTmin" – Lower Boundary for Temperature Interval

Table 3.53 Register `adcTmin`

Name	Address	Bits	Default	Access	Description
adcTmin	$4D_{HEX}$	[7:0]	00_{HEX}	RW	Lower boundary for the temperature interval compared to the upper bits of `adcTdat`

3.8.3.29. Miscellaneous ADC Registers

The registers defined in the next three sections provide settings that enable interrupts or control various functions related to the ADCs.

3.8.3.30. Register "adcAcmp" – ADC Function Enable Register

Table 3.54 Register adcAcmp

Name	Address	Bits	Default	Access	Description
anaGndSw	$4E_{HEX}$	[0]	0_{BIN}	RW	If set to 1, the signal `pdExtTemp` (see Figure 3.12), which is normally controlled by the PMU, is forced to 1. In this case, the transistor shown in Figure 3.12 is not conducting.
ctcvMode		[2:1]	00_{BIN}	RW	Current threshold comparator mode: 00 – The Current Threshold Comparator Mode is disabled. 01 – `adcCtcv` is decremented when the absolute current value is below the threshold and incremented otherwise. 10 – `adcCtcv` is reset when the absolute current value is below the threshold and incremented otherwise. 11 – `adcCtcv` retains its value when the absolute current value is below the threshold and incremented otherwise.
CaccuThEna		[3]	0_{BIN}	RW	If set to 1, enables the strobe to interrupt the controller when the current accumulator exceeds its threshold.
CovrEna		[4]	1_{BIN}	RW	If set to 1, enables the strobes to interrupt the controller when an over-range or overflow has been detected in the current channel.
VTOvrEna		[5]	1_{BIN}	RW	If set to 1, enables the strobes to interrupt the controller when an over-range or overflow has been detected in the voltage/temperature channel.
ctcvRstMode		[6]	0_{BIN}	RW	If set to 1, then `adcCtcv` is reset when the current result counter is reset (`adcCrcv`).
accuRstMode		[7]	0_{BIN}	RW	If set to 1, then `adcCaccu` is reset when the current result counter is reset (`adcCrcv`).
VthWuEna	$4F_{HEX}$	[0]	0_{BIN}	RW	If set to 1, enables the strobe to interrupt the controller for the voltage threshold comparator and voltage accumulator functionality.
VthSel		[1]	0_{BIN}	RW	If set to 0, the absolute value of the single voltage conversion result is compared to the threshold `adcVTh`. If set to 1, the accumulated results of all voltage conversions within an MRCS are compared to the threshold `adcVTh`.
TwuEna		[2]	0_{BIN}	RW	If set to 1, enables the strobe to interrupt the controller for checking the temperature limits.
Unused		[7:3]	00000_{BIN}	RO	Unused; always write as 0.

3.8.3.31. Register “adcGomd” – Reference Voltage and SDM Configuration

Table 3.55 Register adcGomd

Name	Address	Bits	Default	Access	Description
vrefSel	50_{HEX}	[1:0]	00_{BIN}	RW	Selection of the voltage reference: 00_{BIN} vbgh (high precision bandgap) 01_{BIN} vbgl (low power bandgap) 10_{BIN} vcm (common mode voltage) 11_{BIN} External reference voltage
sdmChopClkDiv		[3:2]	00_{BIN}	RW	Divider value for the chop clock related to the SDM clock. See equation (5) in section 3.8.1.1 for the FP state and equation (8) in section 3.8.1.2 for the LP/ULP state.
sdmSetup		[7:4]	0001_{BIN}	RW	Configuration of the initial setup procedure: 0000_{BIN} Execute 4 SDM clock cycles 0001_{BIN} Execute 8 SDM clock cycles … 0111_{BIN} Execute 512 SDM clock cycles 1000_{BIN} to 1111_{BIN} Execute 1024 SDM clock cycles

3.8.3.32. Register “adcSamp” – Oversampling and Filter Configuration

Table 3.56 Register adcSamp

Name	Address	Bits	Default	Access	Description
osr	51_{HEX}	[1:0]	00_{BIN}	RW	Oversampling rate: 00_{BIN} 0 256x oversampling 01_{BIN} 1 128x oversampling 10_{BIN} 2 64x oversampling 11_{BIN} 3 32x oversampling
Unused		[2]	0_{BIN}	RO	Unused; always write as 0.
avgFiltCfg		[4:3]	00_{BIN}	RW	Configuration of post filter (averaging filter) : 00_{BIN} 01_{BIN} No averaging 10_{BIN} 2-stage averaging filter 11_{BIN} 3-stage averaging filter
chopPause		[5]	0_{BIN}	RW	Length of pause in chopping mode: 0 8 SDM clock cycles 1 16 SDM clock cycles
chopShiftPhaseEna		[6]	0_{BIN}	RW	0: Chopping clock is at positive phase of modulator clock 1: Chopping clock is shifted to negative phase of modulator clock
Unused		[7]	0_{BIN}	RO	Unused; always write as 0.

3.8.4. ADC Control and Conversion Timing

In the FP state, the ADC unit is running with the 4 MHz clock derived from the HP oscillator. Its operation is fully controlled by the MCU via register settings. In the LP or ULP state, the ADC unit is running with the 125 kHz clock from the LP oscillator. While basic configurations for the ADC unit are taken from the register file, its operation is fully controlled by the PMU.

3.8.4.1. ADC Operation in the FP State

Before any of the ADCs can be used in the FP state, they must be powered up by setting the `pwrAdcI` bit for the current ADC and/or the `pwrAdcV` bit for the voltage/temperature ADC in the `pwrCfgFp` register to 1 (see Table 3.18). These bits can be kept set to 1 when entering one of the power-down states as the PMU takes over the control of the power signals.

The user can select which kind of operation will be performed by the ADCs. For this, the user can control the input multiplexers shown in Figure 3.12 by setting the field `adcMode` in the `adcCtrl` register appropriately (see Table 3.58). The following settings are possible:

Table 3.57 adcMode Settings

adcMode	Current ADC Configuration	Voltage / Temperature ADC Configuration
0	Current	Voltage
	INP/INN	Divided VBAT/VSSA
1	Current	External temperature
	INP/INN	VDDA/NTH and NTH/NTL
2	Current	Internal temperature
	INP/INN	VPTAT/VREF
3	Offset Calibration Mode; shortened inputs	
	VCM/VCM	VCM/VCM
4	Gain Calibration Mode at maximum (positive) input	
	VREF / VSSA *	VREF/VSSA
5	Gain Calibration Mode at minimum (negative) input	
	VSSA / VREF *	VSSA / VREF
6	1 mV internal test voltage	Voltage
	1 mV/VSSA	Divided VBAT/VSSA
7	Test Mode; each multiplexer is individually controlled by the following:	
	`cSel` field in `adcChan` register	`vtSel` field in `adcChan` register

*** Note:** The two Gain Calibration Modes cause an ADC over-range error in the current ADC as the minimum gain of PGA2 is 4. Therefore these modes are not usable for the current ADC.

After setting the desired mode of operation, the user must start the conversion by setting the `startAdcC` bit in the `adcCtrl` register (see Table 3.58) for the current channel and/or the `startAdcV` bit for the voltage/temperature channel to 1. After an initial setup phase, measurement results are stored in the corresponding result registers. By controlling the `startAdc` bits, the user is able to generate an individual conversion sequence (ADC operation stops after one conversion sequence has finished) or continuous conversion (ADC operation continues after one conversion sequence has finished).

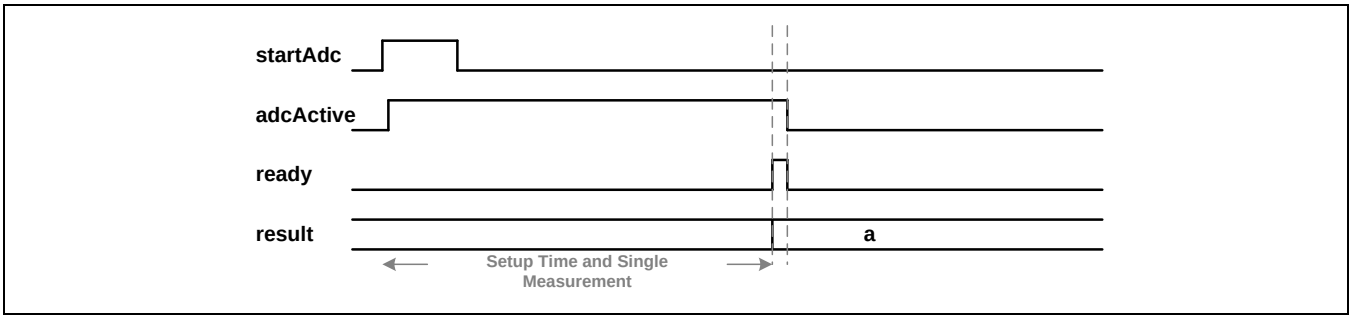
© 2016 Integrated Device Technology, Inc.

A conversion sequence is defined as a series of several measurements. The number of measurements to be performed is controlled by the result counter functionality, so it is possible to have multiple measurements per conversion sequence (MRCS) or just a single measurement (SRCS). At the end of one conversion sequence, the "set interrupt" strobe for the corresponding conversion interrupt ready status bit (`irqStat[7:5]`) is generated. Although this strobe is only generated after the last measurement within an MRCS, each measurement in the MRCS is used for accumulation and min/max determination.

Note: The MRCS functionality is only available for current and voltage measurements, not for temperature measurements.

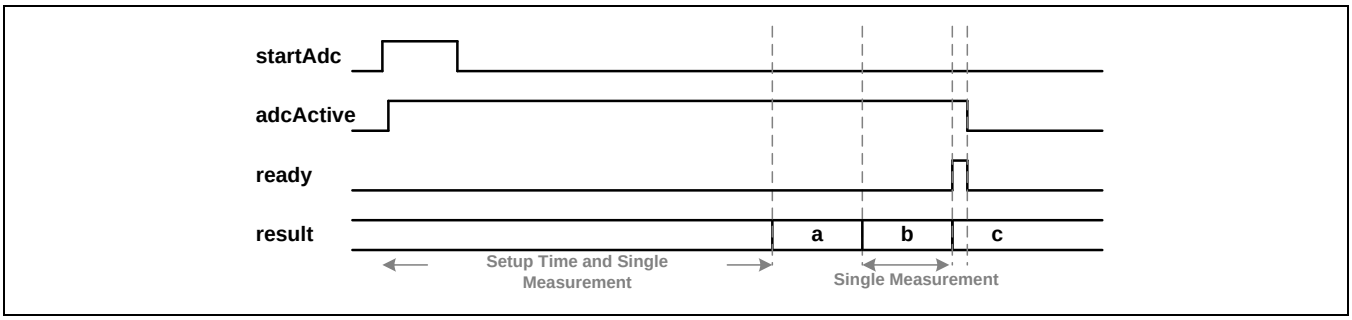
To perform an individual conversion sequence for SRCS or MRCS, the user must generate a strobe signal on the corresponding `startAdc` bit by setting the `startAdc` bit to 1 (rising edge) first and then to 0 (falling edge). The rising edge of `startAdc` signals the ADC to start the conversion. On this start signal, the corresponding `adcActive` flag is set to 1, which can be read from bits 4 and/or 5 in the SSW. When the conversion sequence has finished, the corresponding ready signal is generated. At that time, the internal logic evaluates the status of the `startAdc` bit again. If it was cleared already as required for an individual conversion sequence, the ADC stops its operation and clears the `adcActive` flag. This behavior is shown in Figure 3.22 and Figure 3.23.

Figure 3.22 Individual SRCS



Note that the ADC stops since `startAdc` is low at the end of the conversion sequence for the individual SRCS.

Figure 3.23 Individual MRCS (Example for Result Counter of 3)

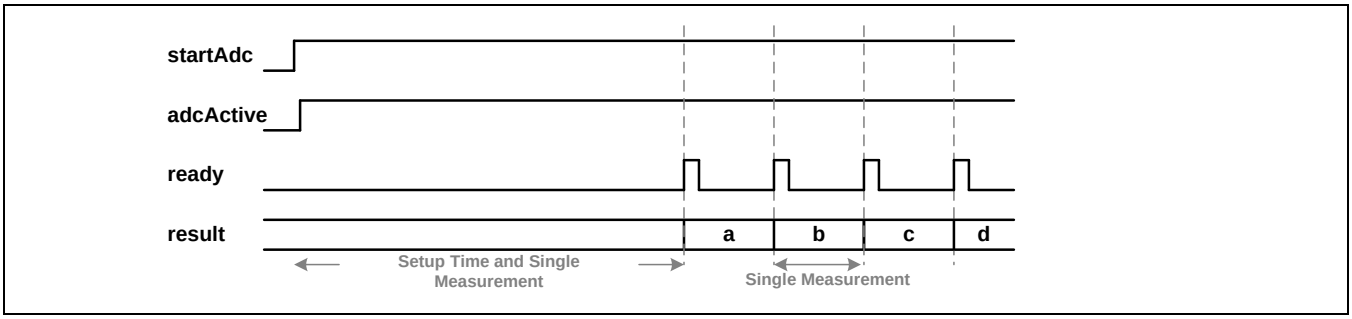


Note that the ADC stops since `startAdc` is low at the end of the conversion sequence for an individual MRCS.

Important: The ready strobe shown in Figure 3.22 and Figure 3.23 is used to set the interrupt status bit, but the interrupt status bit remains set until it is cleared by software.

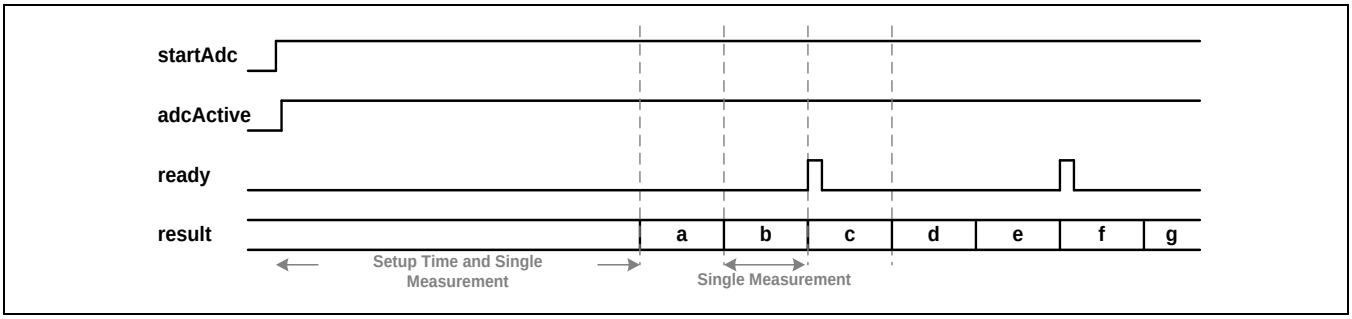
To perform a continuous conversion sequence for SRCS or MRCS, the user must set the corresponding `startAdc` bit to 1 (rising edge). The rising edge of `startAdc` signals the ADC to start the conversion. On this start signal, the corresponding `adcActive` flag is set to 1, which can be read from bits 4 and 5 in the SSW. When one conversion sequence has finished, the corresponding ready signal is generated. At that time, the internal logic evaluates the status of the `startAdc` bit again. As the `startAdc` bit is still 1, the ADC continues its operation but without the need for the setup time. This behavior is shown in Figure 3.24 and Figure 3.25.

Figure 3.24 Continuous SRCS



Note that the ADC continues since `startAdc` is high at the end of the conversion sequence for a continuous SRCS.

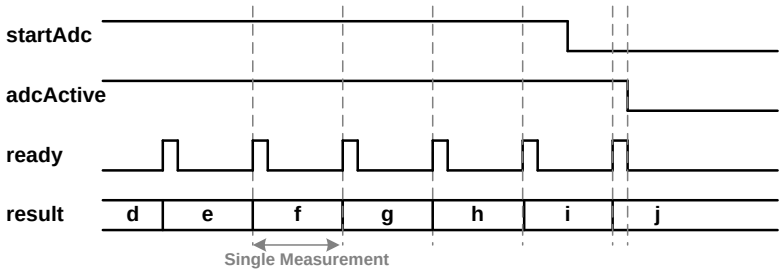
Figure 3.25 Continuous MRCS (Example for Result Counter of 3)



Note that the ADC continues since `startAdc` is high at the end of the conversion sequence for a continuous MRCS.

When a continuous conversion sequence is performed that will be stopped after the present active conversion sequence has completed, the user only needs to clear the `startAdc` bit of the channel that will be stopped. Then the user can either wait for the next interrupt, which will be set by the last ready strobe, or check the corresponding `adcActive` bit in the SSW.

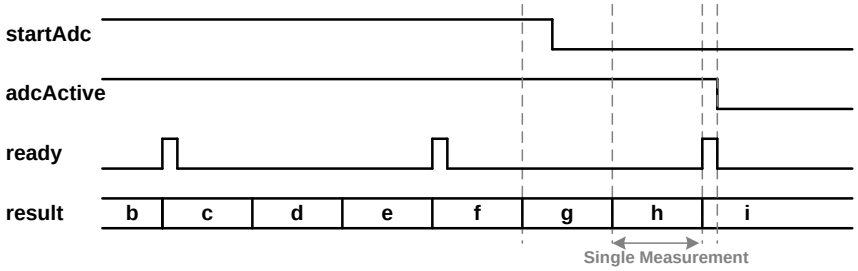
Figure 3.26 Stopping Continuous SRCS



Note that the ADC stops since `startAdc` is low at the end of the conversion sequence for a continuous SRCS.

When a conversion sequence is performed that will be interrupted (stopped immediately), the user must clear the `startAdc` bit of the channel that will be stopped (when set) and must set the `stopAdc` bit in the `adcCtrl` register to 1. In the ADC unit, the `stopAdc` bit is only evaluated when the `startAdc` bit of a channel is low and the corresponding `adcActive` bit is high.

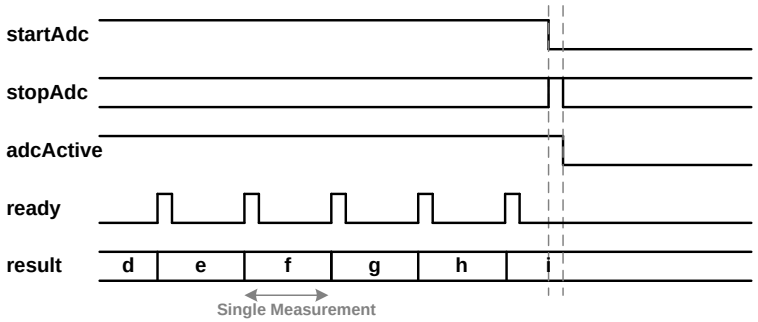
Figure 3.27 Stopping Continuous MRCS (Example for Result Counter of 3)



Note that the ADC stops since `startAdc` is low at the end of the conversion sequence for a continuous MRCS; otherwise the `stopAdc` bit is ignored. Therefore there is only one `stopAdc` bit that is used for both channels. This allows the user to stop both channels by clearing both `startAdc` bits when setting the `stopAdc` bit or to stop only one channel by keeping one `startAdc` bit high.

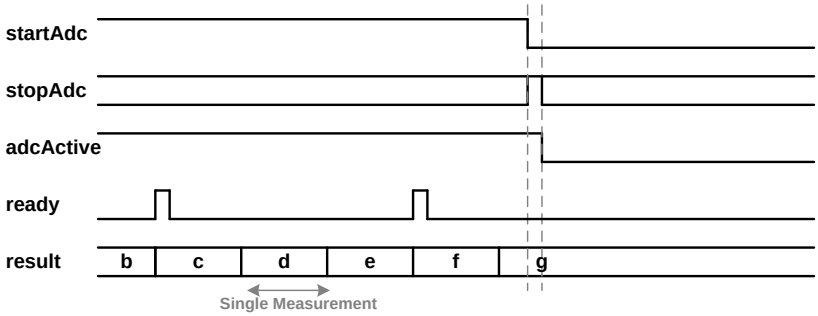
The signal behavior for interrupting a channel is shown in Figure 3.28 and Figure 3.29.

Figure 3.28 Interrupting a Continuous SRCS



Note that the ADC immediately stops since `startAdc` is low and `stopAdc` is high.

Figure 3.29 Interrupting a Continuous MRCS (Example for Result Counter of 3)



Note that the ADC immediately stops since `startAdc` is low and `stopAdc` is high.

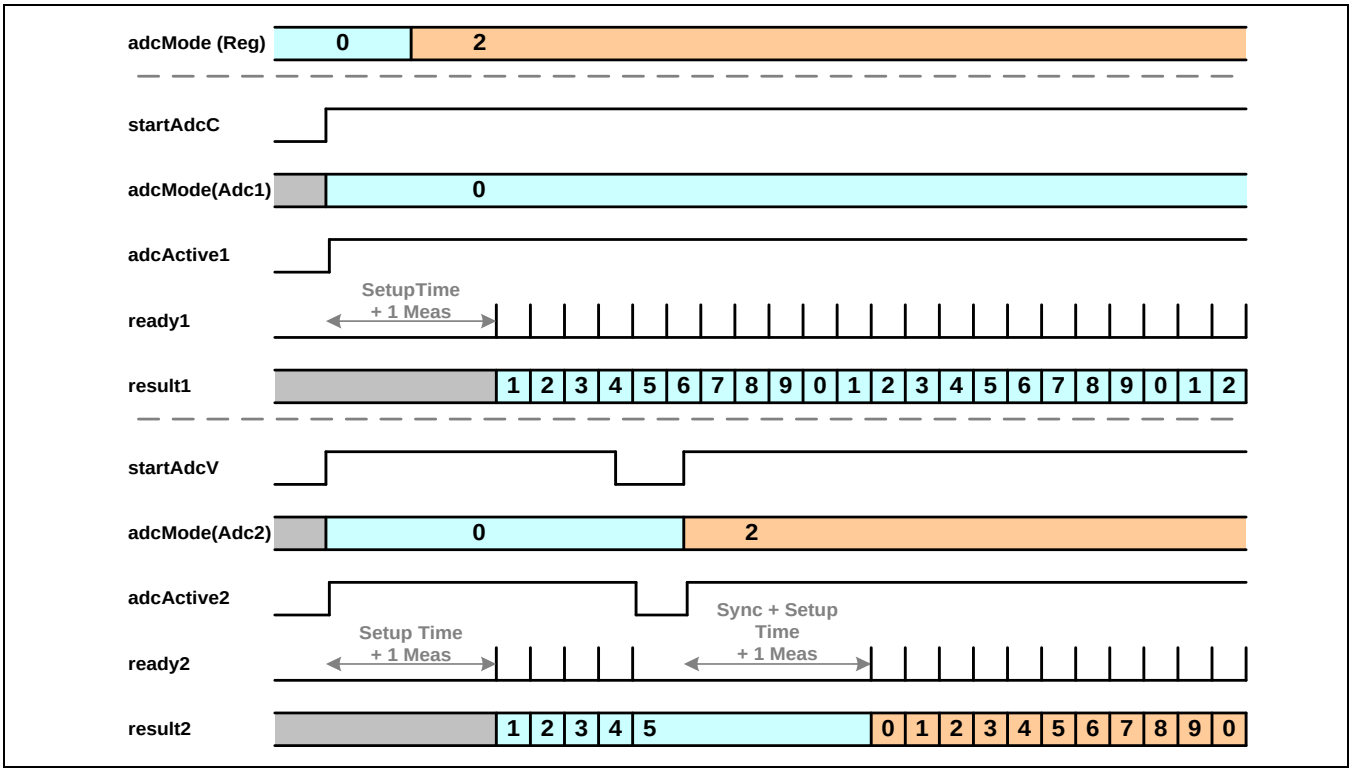
Important: The `stopAdc` bit is only evaluated when `startAdc` bit is low.

Note: The interrupt sequence shown in Figure 3.28 and Figure 3.29 is also performed by the PMU on transition from the FP state to any power-down state as well as on transition from any power-down state to the FP state. This allows the user to keep the `startAdc` bits set on transition to any power-down state. After wake-up, the ADCs continue the operation they performed before going to power-down.

Most of the register settings that influence both ADC channels (e.g., oversampling rate) can only be changed when both ADC channels are inactive. As explained above, this is not true for the `stopAdc` bit. The `adcMode` field can also be changed while any ADC channel is active. This is useful for continuing with current measurements in the first ADC channel while changing the second ADC channel from voltage to temperature measurements (as an example). On the rising edge of its `startAdc` bit, each ADC channel stores internally the mode it is configured for and keeps this setting until the next rising edge of its `startAdc` bit.

When one channel is reconfigured while the other one is active, this channel does not start immediately after being re-enabled but synchronizes to the active channel so that the results are generated at the same time. This is shown in Figure 3.30.

Figure 3.30 Signal Behavior of adcMode



3.8.4.2. Register “adcCtrl” – ADC Control Register

Table 3.58 Register `adcCtrl`

Name	Address	Bits	Default	Access	Description
startAdcC	56_{HEX}	[0]	0_{BIN}	RW	Start signal for the current ADC; used in the FP state, ignored in other states.
startAdcV		[1]	0_{BIN}	RW	Start signal for the voltage ADC; used in the FP state, ignored in other states.
stopAdc		[2]	0_{BIN}	RW	Stop signal for both ADCs; used in the FP state, ignored in other states.
adcMode		[5:3]	000_{BIN}	RW	ADC multiplexer configuration; used in the FP state, ignored in other states; for settings 0, 1, 2, and 6, the first value is applied to the current ADC, the second to the voltage ADC. 0 – Measure current and voltage 1 – Measure current and external temperature 2 – Measure current and internal temperature 3 – Offset calibration 4 – Gain calibration at maximum (positive) input 5 – Gain calibration at minimum (negative) input 6 – Internal test voltage and voltage 7 – Test Mode (control multiplexer via the `adcChan` register’s `cSel` and `vtSel` fields)
chopEna		[6]	0_{BIN}	RW	If set to 1, Chopping Mode is enabled.
Unused		[7]	0_{BIN}	RO	Unused; always write as 0.

3.8.4.3. ADC Operation in LP / ULP State

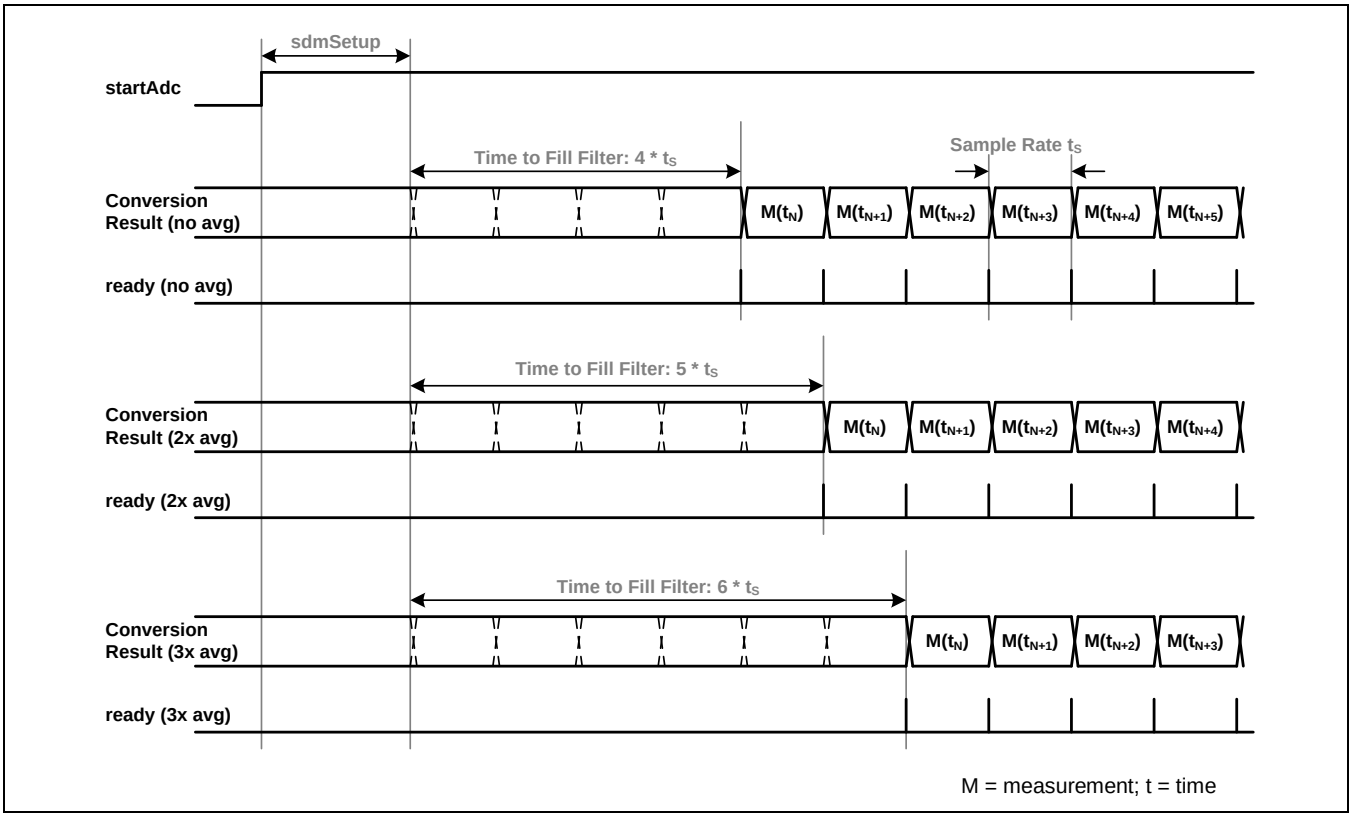
During the LP or ULP state, the ADCs are fully controlled by the PMU depending on the settings of register `pwrCfgLp` (see Table 3.19). The PMU overrides the settings of the `startAdc` bits, the `stopAdc` bit, and `adcMode` field. The settings of `pwrAdcI` and `pwrAdcV` are also ignored until the system wakes up. While no further settings are required for the continuous measurement set-ups, the user can independently configure how many current and/or voltage measurements happen within a single measurement window. For current (the green and orange boxes In Figure 3.6 to Figure 3.9), the number of current measurements in each window is configured by the setting of `adcCrcl`. For voltage (the orange boxes in Figure 3.7 through Figure 3.9), the number of voltage measurements in each window is configured by the setting of `adcVrcl`. There is always only one temperature measurement.

Important: If an interrupt wakes up the system before the end of a measurement window, the conversion sequence is interrupted and less than the configured number of measurements will have been completed. This can be checked by the registers `adcCrcv` and `adcVrcv`.

3.8.4.4. ADC Conversion Timing

The complete conversion process is controlled by an internal state machine that guarantees that only valid measurement results are used. The base for all ADC timings is the SDM clock, which is generated from the 4MHz clock in FP state or from the 125kHz clock in LP and ULP state. After the ADC measurement has been started (rising edge of `startAdc`), the state machine always introduces a configurable number of SDM clock cycles (field `sdmSetup` in register `adcGomd`) to allow the analog part of the SDM to settle. After this delay, the incoming bit streams are used to fill the sinc^4 decimation filter. This lasts 4 times the sample rate, which is configured by the oversampling rate (the `osr` field in register `adcSamp`). Then the first valid result value comes from the decimation filter.

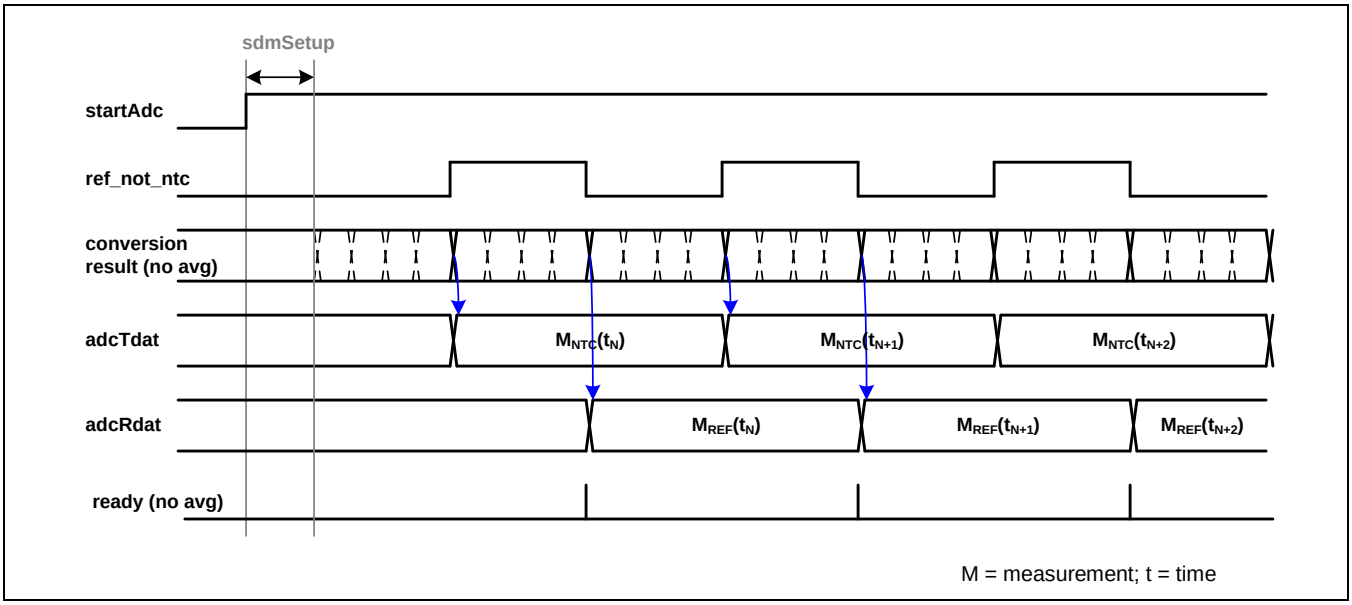
Figure 3.31 Timing for Current, Voltage, and Internal Temperature Measurements without Chopping for Different Configurations of the Average Filter



For current, voltage, or internal temperature measurement without chopping, when only one input source must be measured, this is the first valid value. The time when the first valid result is present also depends on the configuration of the average filter (the `avgFiltCfg` field in register `adcSamp`). If no averaging is used, the first valid value is also the first valid result stored in `adcCdat`, `adcVdat`, or `adcTdat`. For the 2-stage or 3-stage average filter, respectively, two or three valid values are needed to calculate a valid result. This adds an additional delay, respectively, of 1 or 2 times the sample rate.

For external temperature measurement without chopping, two input sources must be measured: the voltage drop over the reference resistor (the result is stored in register `adcRdat`) and the voltage drop over the NTC resistor (result stored in register `adcTdat`). The `sdmSetup` time is only introduced at the beginning of the conversion sequence (the rising edge of `startAdc`). Each single measurement of one of the two values needs 4 times the sample rate when averaging is disabled, or respectively, 5 or 6 times the sample rate when using the 2-stage or 3-stage average filter. This also means that a complete pair of values used to calculate one external temperature value needs 8 (10 or 12 for averaging) times the sample rate because for each value, the pipeline of the sinc^4 decimation filter must be filled first.

Figure 3.32 Timing for External Temperature Measurements without Chopping when No Average Filter is Enabled



Note that using an average filter will lead, respectively, to 5 and 6 conversion results during each high and low phase of ref_not_ntc.

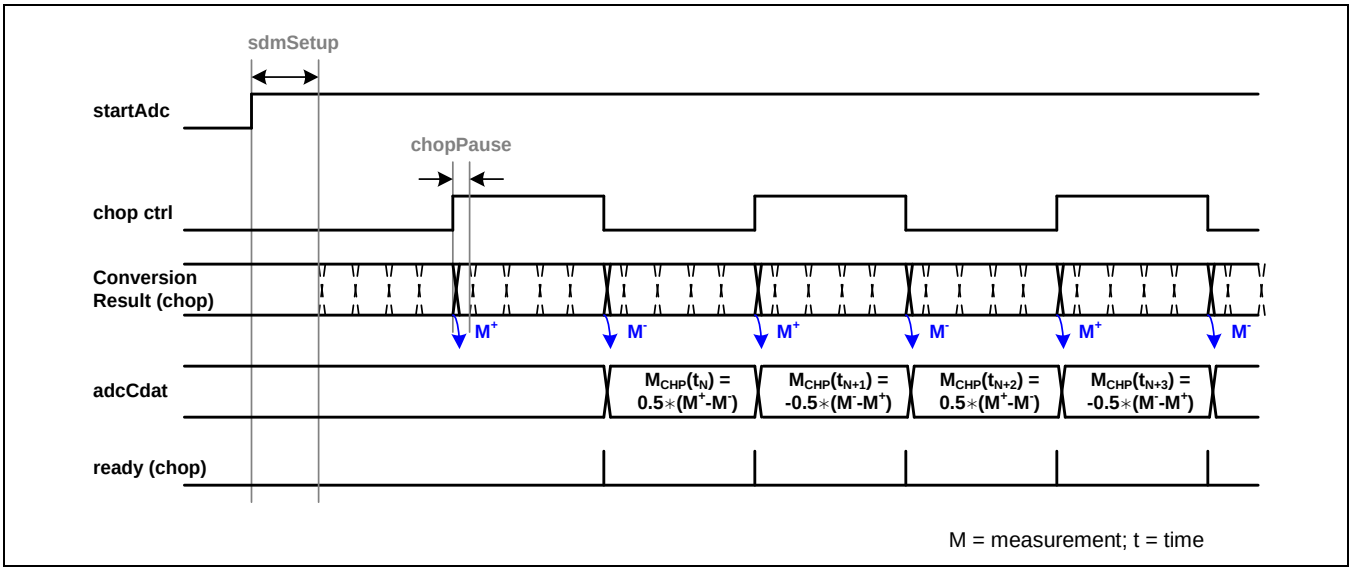
The timings shown in the previous two figures are without chopping, which means that the differential input signal is always applied in the same manner to the analog SDM-ADC. Although this kind of measurement is fast (one result value after each sample time), it has the drawback that it also converts any offset present in the analog blocks. This would lead to less accurate measurement results. To overcome this, chopping can be enabled (bit `chopEna` in register `adcCtrl`). When chopping is enabled, the differential input signal is directly applied to the analog SDM-ADC the first time and inverted the second time. Taking this into account in the digital part removes the offset applied by the ADC itself:

$$\text{data} = \frac{(V_{in} + \text{offset}) + (-1) * (-V_{in} + \text{offset})}{2} = V_{in} \quad (15)$$

For current, voltage, or internal temperature measurement with Chopping Mode enabled (chopEna set to 1), this leads to a timing similar to the external temperature measurement without chopping and averaging since two values are measured: the normal input and the inverted input. Each single measurement of one of the two values needs 4 times the sample rate as no averaging of the single measurement is performed. Instead, the average filter is automatically configured as a 2-stage average filter to calculate the formula above.

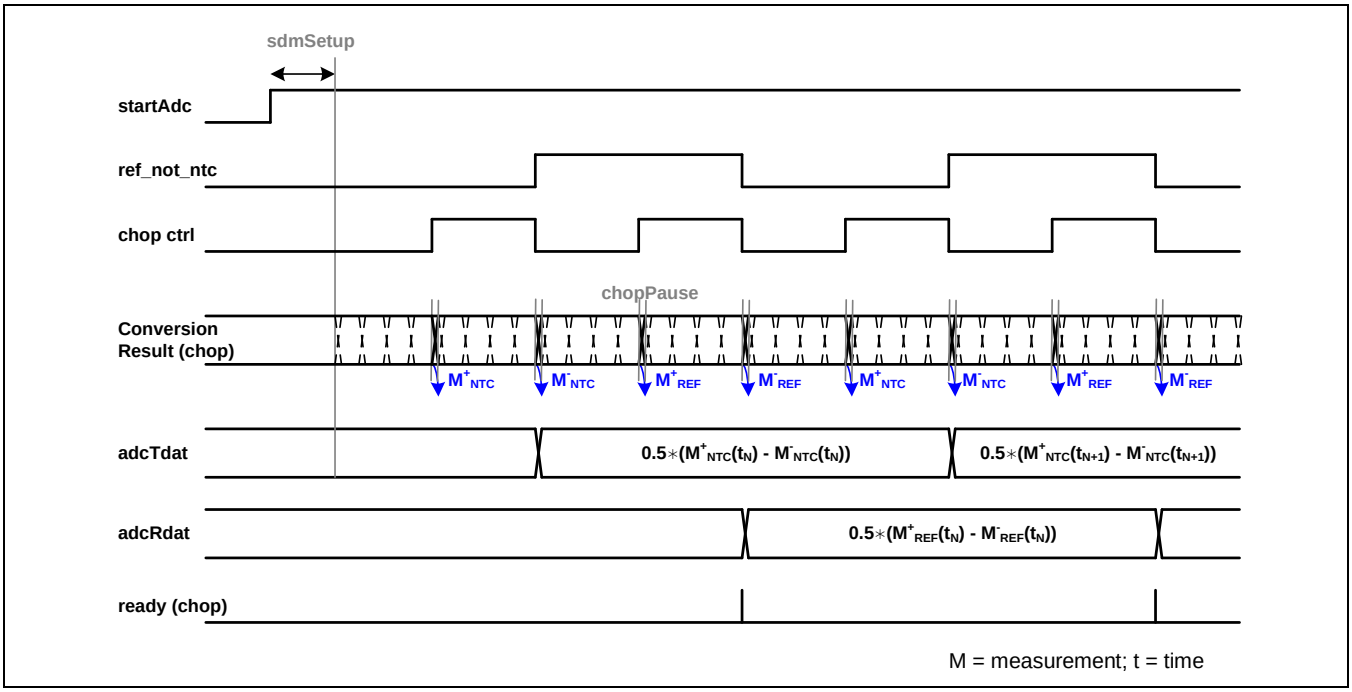
The second difference is that a small pause (chopping pause) is introduced each time the chop control signal changes to allow the analog blocks to settle due to the input change. This is possible since the chopEna bit influences both ADC paths. The length of the chop pause is either 8 or 16 SDM clock cycles, which can be configured using the chopPause bit in register adcSamp.

Figure 3.33 Timing for Current, Voltage, and Internal Temperature Measurements using Chopping – Example Showing Current (adcCdat)



For external temperature measurement using chopping, two different input sources must be measured twice, non-inverted and inverted, which leads to four values to be measured to get a result. To keep both ADC paths aligned, the chopPause is introduced for each measured value.

Figure 3.34 Timing for External Temperature Measurements using Chopping



Important: The timings only show the principle. Additional small delays such as pipeline delays are not included.

3.8.5. Diagnostic Features

3.8.5.1. ADC Analog Multiplexer Control for Diagnosis and Test

In the FP state, the three multiplexers shown in Figure 3.12 can be directly controlled via register `adcChan` when the `adcMode` field in register `adcCtrl` is set to 7 (see Table 3.58). For other settings of `adcMode`, the settings of register `adcChan` are ignored and both multiplexers for input selection are controlled either by the `adcMode` field in FP state or by the PMU in LP or ULP state.

The `vtSel` field in register `adcChan` is used to select the input sources of the voltage/temperature ADC. The `cSel` field in register `adcChan` is used to select the input sources of the current ADC. See Table 3.59.

Note: The reference voltage (non-inverted as well as inverted) cannot be measured by the current ADC as the minimum gain of PGA2 is 4, which causes an ADC overrange error.

For some settings of `adcMode`, `cSel` and `vtSel`, the reference voltage is applied to the ADCs. The user can select the source of the reference voltage by field `vrefSel` in register `adcGomd` (see section 3.8.3.31). As can be seen from in Figure 3.12, the user can connect internal current sources to the input wires of INP and INN as well as to the input wires of NTH and NTL. To enable the different current sources for the four input wires, the corresponding enable bit in register `currentSrcEna` must be set to 1 (see Table 3.62).

Important Warning: Do not enable both current sources on the same input at once.

Important: The current sources can be enabled independent of the `adcMode`.

3.8.5.2. Register “adcChan” – Analog Multiplexer Configuration

Table 3.59 Register adcChan

<table>
<tr><th>Name</th><th>Address</th><th>Bits</th><th>Default</th><th>Access</th><th>Description</th></tr>
<tr><td>vtSel</td><td rowspan="4">D0$_{HEX}$</td><td>[2:0]</td><td>000$_{BIN}$</td><td>RW</td><td>When adcMode == 7, this field selects the differential sources for the voltage/temperature ADC:
<table>
<tr><th>vtSel</th><th>inp</th><th>inn</th></tr>
<tr><td>000$_{BIN}$</td><td>VDDA</td><td>NTH</td></tr>
<tr><td>001$_{BIN}$</td><td>NTL</td><td>NTH</td></tr>
<tr><td>010$_{BIN}$</td><td>VPTAT</td><td>VBGH (i.e., V_{REF})</td></tr>
<tr><td>011$_{BIN}$</td><td>VBATP</td><td>VBATN</td></tr>
<tr><td>100$_{BIN}$</td><td>VBGH (i.e., V_{REF})</td><td>VSSA</td></tr>
<tr><td>101$_{BIN}$</td><td>VBGL (i.e., V_{REFLP})</td><td>VSSA</td></tr>
<tr><td>110$_{BIN}$</td><td>VCM</td><td>VCM</td></tr>
<tr><td>111$_{BIN}$</td><td>High impedance</td><td>High impedance</td></tr>
</table></td></tr>
<tr><td>cSel</td><td>[5:3]</td><td>000$_{BIN}$</td><td>RW</td><td>When adcMode == 7, this field selects the differential sources for the current ADC:
<table>
<tr><td>000$_{BIN}$</td><td>INP</td><td>INN</td></tr>
<tr><td>001$_{BIN}$</td><td>INP</td><td>INN</td></tr>
<tr><td>010$_{BIN}$</td><td>INP</td><td>INN</td></tr>
<tr><td>011$_{BIN}$</td><td>INP</td><td>INN</td></tr>
<tr><td>100$_{BIN}$</td><td>1 mV</td><td>VSSA</td></tr>
<tr><td>101$_{BIN}$</td><td colspan="2">Unused</td></tr>
<tr><td>110$_{BIN}$</td><td colspan="2">Unused</td></tr>
<tr><td>111$_{BIN}$</td><td>VCM</td><td>VCM</td></tr>
</table></td></tr>
<tr><td>Unused</td><td>[6]</td><td>0$_{BIN}$</td><td>RW</td><td>Unused; always write as 0</td></tr>
<tr><td>Unused</td><td>[7]</td><td>0$_{BIN}$</td><td>RW</td><td>Unused; always write as 0.</td></tr>
</table>

3.8.6. Digital Test Features

3.8.6.1. Built-in Self-Test (BIST)

The digital ADC BIST feature allows the user to test the digital logic of the ADC data path. The BIST feature is enabled by setting the bistEna bit in the adcDiag register to 1 (see Table 3.61). When the BIST feature is enabled, the same programmable bit stream is applied to both inputs of the decimation filter instead of the outputs from the noise cancellation filters. The ADCs must also be set into operation as during normal operation.

The bit stream to be applied to the decimation filter is programmed to the lower 30 bits of register adcCaccTh. These 30 bits function as a shift-rotate register as shown in Figure 3.35, and the output of the lowest bit is used as the bit stream for the BIST.

Important: When register adcCaccTh is used for the BIST, the current accumulator threshold functionality cannot be used.

Figure 3.35 Using Register adcCaccTh for the Digital ADC BIST

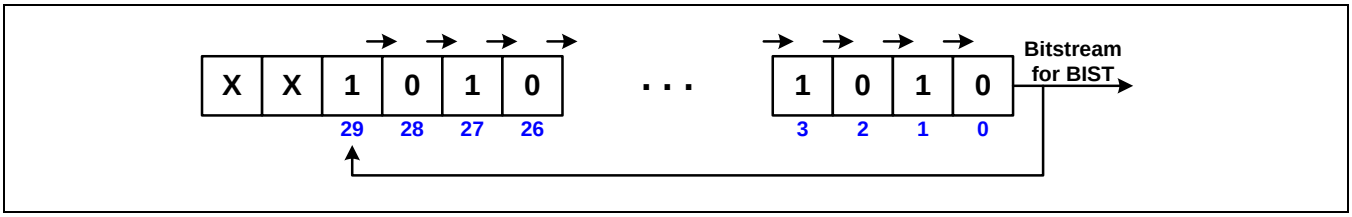


Table 3.60 shows four example bit streams as well as the expected output stored in the corresponding data registers `adcCdat`, `adcVdat`, `adcTdat`, and/or `adcRdat` if enabled. In these examples, the offset correction value (e.g., register `adcCoff`) is set to 0; the gain correction value (e.g., register `adcCgan`) is set to 1.0; and the post correction gain factor (e.g., bit field `curPoCoGain` in register `adcPoCoGain`) is set to gain factor 1 (bit field set to 00_{BIN}) and then to gain factor 2 (bit field set to 01_{BIN}).

Table 3.60 Example Results of BIST

1/0 Bit Ratio	Bit Stream	Result Data (xPoCoGain = Gain Factor 1; Bit Field = 00_{BIN})	Result Data (xPoCoGain = Gain Factor 2; Bit Field = 01_{BIN})
1/6	100000_100000_100000_100000_100000 (20820820_{HEX})	$AAAAAA_{HEX}$	800000_{HEX} (negative over-range)
5/6	111110_111110_111110_111110_111110 ($3EFBEFBE_{HEX}$)	555555_{HEX}	$7FFFFF_{HEX}$ (positive over-range)
2/5	10010_10010_10010_10010_10010_10010 ($25294A52_{HEX}$)	$E66666_{HEX}$	$CCCCCC_{HEX}$
3/5	10110_10110_10110_10110_10110_10110 ($2D6B5AD6_{HEX}$)	199999_{HEX}	333332_{HEX}

3.8.6.2. Decimation Filter Output Test

The decimation filter output test allows the user to observe the outputs of both decimation filters. This feature is enabled by setting bit `rawEna` in register `adcDiag` to 1 (see Table 3.61). When this feature is enabled, the 32-bit output value of the decimation filter for the current ADC is stored in registers `adcCmax` (MSBs; see Table 3.48) and `adcCmin` (LSBs; see Table 3.49) and the 32 bit output value of the decimation filter for the voltage/temperature ADC is stored in registers `adcVmax` (MSBs; see Table 3.50) and `adcVmin` (LSBs; see Table 3.51). The ADCs must also be set into operation as during normal operation.

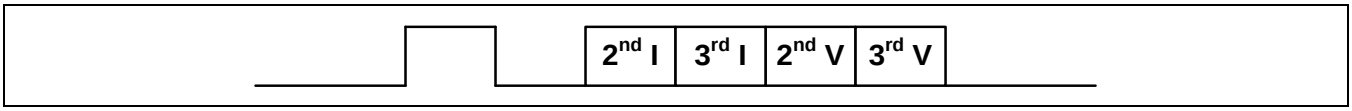
Note: When this feature is enabled, all normal ADC operations described in the previous sections function as described except the minimum and maximum functionality for the current and voltage values as the registers are used for this test function.

Note: This feature can be combined with the digital ADC BIST feature.

3.8.6.3. ADC Interface Test

The ADC interface test allows the user to observe the incoming 2nd and 3rd order bit streams from both analog parts of the SD-ADCs. This feature is enabled by setting bit `adcIfTestEna` in register `adcDiag` to 1. The digital part of the ADC unit must be enabled as for normal operation as it generates the correct sample strobe for the test logic. This function is only available in the FP state as it runs on the 20MHz clock from the high-precision oscillator. All sampled values (4 bits) are shifted out of the STO pad. To enable the user to synchronize on the sampled data, a 1 and a 0 are shifted out before each 4-bit value as shown in Table 3.37.

Figure 3.36 Bit Stream of ADC Interface Test at STO Pad



Note: This feature can be combined with the digital ADC BIST feature as well as with the decimation filter output test.

3.8.6.4. Register "adcDiag" – Enable Register for Test and Diagnosis Features

Table 3.61 Register `adcDiag`

Name	Address	Bits	Default	Access	Description
bistEna	D1$_{HEX}$	[0]	0_{BIN}	RW	If set to 1, enables BIST.
rawEna		[1]	0_{BIN}	RW	If set to 1, enables the decimation filter output test (ADC raw data test).
adcIfTestEna		[2]	0_{BIN}	RW	If set to 1, enables the serial ADC test.
Unused		[5:3]	000_{BIN}	RW	Unused; always write as 0.
stopClkChop		[6]	0_{BIN}	RW	Disable signal for the chop clock generation in the digital section.
clkChopEna		[7]	1_{BIN}	RW	Enable signal for the chop clock used partially in the analog section.

3.8.6.5. Register "currentSrcEna" – Enable Register for Current Source

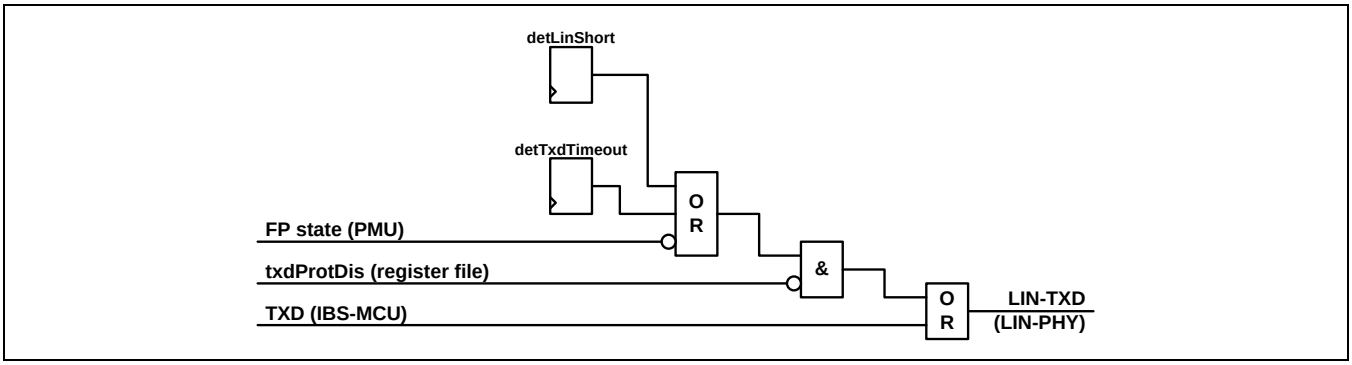
Table 3.62 Register `currentSrcEna`

Name	Address	Bits	Default	Access	Description
inampInpSrcEna	D2$_{HEX}$	[0]	0_{BIN}	RW	Enable 50µA current source to INP
Unused		[1]	0_{BIN}	RW	Unused; always write as 0
inampInnSrcEna		[2]	0_{BIN}	RW	Enable 50µA current source to INN
Unused		[3]	0_{BIN}	RW	Unused; always write as 0
psrcEnVbat		[4]	0_{BIN}	RW	Enable 50µA current source on NTH
psinkEnVbat		[5]	0_{BIN}	RW	Enable -50µA current source on NTH
nsrcEnVbat		[6]	0_{BIN}	RW	Enable 50µA current source on NTL
nsinkEnVbat		[7]	0_{BIN}	RW	Enable -50µA current source on NTL

3.9. SBC LIN Support Logic

The LIN support logic contains only two functions – most of the LIN communication is handled by the LIN PHY on one side and the MCU on the other side (see section 4.7). The two functions are used to handle error conditions (TXD timeout; LIN short) and protect (force to high level) the LIN TXD line to LIN PHY if any of these errors is detected or when the system is not in full-power state. Figure 3.37 illustrates the error protection logic discussed in the next sections.

Figure 3.37 Protection Logic of the LIN TXD Line



3.9.1. LIN Wakeup Detection

A LIN master generates a LIN wakeup frame by driving a dominant value of 0 of at least 250µs on the LIN bus. The standard requires that a LIN slave shall recognize a LIN wakeup when the LIN bus is low for more than 150µs.

There is a 6-bit counter running with the 125kHz LP clock implemented in the IBS-SBC to support the LIN wakeup detection. When the function is disabled (`irqEn[4]` is set to 0), the counter is set to 20_{HEX} and no interrupt can occur. When the function is enabled (`irqEn[4]` is set to 1), the LIN RXD line is observed. When the LIN RXD line is high, the counter is set to 00_{HEX}. When the LIN RXD line becomes low, the counter is incremented in each clock cycle until it reaches the value 20_{HEX} where it stops incrementing. When the counter is equal to the programmed wakeup delay (register `linWuDelay`; see Table 3.66), a set strobe for the corresponding interrupt is generated, which causes the system to wake up.

The register `linWuDelay` has a default value of 14_{HEX}. This setting guarantees that no low level less than 150µs on the LIN RXD line causes a wakeup due to the inaccuracy of the LP oscillator.

3.9.2. TXD Timeout Detection

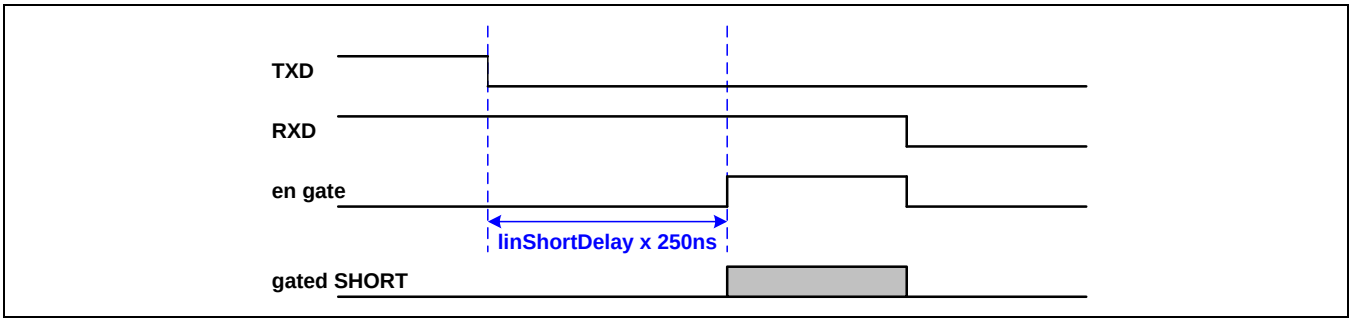
The digital LIN controller in the MCU ensures that it does not completely block the LIN bus due to continuously transmitting a dominant value of 0. As it is still possible that the TXD line from the MCU could be stuck at 0 due to a software or hardware error in the MCU or a broken connection between the two chips, the LIN support logic observes the TXD line in full-power (FP) state to detect if the TXD line is erroneously low. Because the digital LIN controller of the MCU is built for baud rates down to 1kBaud, the maximum time that the digital LIN controller (slave device!) is transmitting a low level is 9ms (start bit and 8 data bits). To overcome inaccuracies of the internal clocks, the internal logic and untrimmed LIN nodes, the timeout value is 10.24ms. On detection of a TXD timeout, an internal flag (`detTxdTimeout` in Figure 3.37) is set to the high level, which forces the LIN TXD line to 1, and the corresponding interrupt status (`irqStat[2]`) is set. While the interrupt status bit is cleared on read access to the interrupt status register, the internal flag remains high, also keeping LIN TXD at the high level. The status of the internal flag is mirrored in `SSW[2]`. To clear this internal flag and to be able to transmit again via the LIN bus, a value of 1 must be written to bit `clrTxdTimeout` in register `linCfg` (see Table 3.63).

3.9.3. LIN Short Detection

The LIN PHY contains a function to detect a short to VBAT on the LIN bus by sensing the current through the open-drain output transistor in the LIN PHY. When the current is too high, the LIN PHY drives the SHORT signal going to the digital block to the high level (see Figure 3.38). Under normal circumstances, the LIN PHY signals a short only if a dominant value of 0 will be transmitted, but the bus remains at its recessive high level. However, high current consumption is also possible due to EMC events. To increase the safety of the system and to avoid misinterpretation, the incoming SHORT signal is gated and filtered.

First, the SHORT signal from the LIN PHY is driven through a configurable gating block in the digital block. The gating block is configured using register `linShortDelay` (see Table 3.65). If register `linShortDelay` is set to a value not equal to 0, the TXD line going to and the RXD line coming from the LIN PHY are observed. When the TXD line becomes low while the RXD line remains high, the gating block waits for `linShortDelay` times 4MHz clock cycles before opening the gate. The gate is closed when either TXD becomes high again or RXD becomes low (see Figure 3.38). This feature is used to evaluate the SHORT signal only when a dominant value of 0 is transmitted but the bus remains at its recessive high level as well as to eliminate the delay from the TXD line through the LIN PHY back to the RXD line.

Figure 3.38 Waveform Showing the Gating Principle for Non-zero Values of `linShortDelay`



When the register `linShortDelay` is set to 0, the gate for the SHORT signal is always open. This means that the SHORT signal is always passed through the gating block even when the TXD line is high or the RXD line is low.

The gated SHORT signal is applied to a configurable de-bouncing filter. This de-bouncing filter is configured using register `linShortFilter` (see Table 3.64), and it observes the gated SHORT signal using the internal 4MHz clock. When the gated SHORT signal is continuously high for (`linShortFilter + 1`) clock cycles, the LIN short interrupt status bit (`irqStat[3]`) is set, enabling the software running on the MCU to respond to this situation. The interrupt status bit is cleared on read access to the interrupt status register.

The software can also enable the hardware to protect the TXD line in case of a detected short condition. When the `shortProtEna` bit in register `linCfg` (see Table 3.63) is set to 1 and a short condition is detected by the de-bouncing filter, an internal flag (`detLinShort` in Figure 3.37) is set to the high level, which forces the LIN TXD line high. The status of the internal flag is mirrored in `SSW[3]`. The internal flag remains high until it is explicitly cleared by the software by writing a value of 1 to the `clrLinShort` bit in register `linCfg`.

3.9.4. LIN Testing

The LIN TXD line protection features (TXD timeout, LIN short, low-power (LP) state) might restrict the possibility of testing the LIN PHY. Therefore the protection can be disabled (see Figure 3.7) by setting the `txdProtDis` bit in register `linCfg` to 1.

Important Warning: This must never be done during normal operation (the IC will not be damaged, but communication errors are not detected).

3.9.4.1. Register “linCfg” – LIN Configuration Register

Table 3.63 Register `linCfg`

Name	Address	Bits	Default	Access	Description
linFastEna	$B4_{HEX}$	[0]	0_{BIN}	RW	When set to 1, the slew rate control in the LIN PHY transmitter is disabled allowing higher LIN data rates of up to 125kBaud (non-standard feature).
txdProtDis		[1]	0_{BIN}	RW	When set to 1, all protection features that force the LIN TXD line to 1 are overwritten (for test purposes only).
shortProtEna		[2]	0_{BIN}	RW	If set to 1, enables the LIN short protection.
Unused		[3]	0_{BIN}	RO	Unused; always write as 0.
clrTxdTimeout		[4]	0_{BIN}	RWS	Strobe register; write 1 to clear the detected TXD timeout flag and to release the protection of the LIN TXD line.
clrLinShort		[5]	0_{BIN}	RWS	Strobe register; write 1 to clear the detected LIN SHORT flag and to release the protection of the LIN TXD line.
Unused		[7:6]	00_{BIN}	RO	Unused; always write as 0.

3.9.4.2. Register “linShortFilter” –Configuration Register for the LIN Short De-bounce Filter

Table 3.64 Register `linShortFilter`

Name	Address	Bits	Default	Access	Description
linShortFilter	$B5_{HEX}$	[7:0]	$0F_{HEX}$	RW	Filter configuration for the LIN short detector. This register defines the number of 4MHz clock cycles (`linShortFilter` + 1) where the gated LIN SHORT signal in the LIN PHY must be high to detect a SHORT condition on the LIN bus.

3.9.4.3. Register “linShortDelay” – Configuration Register LIN Short TX-RX Delay

Table 3.65 Register `linShortDelay`

Name	Address	Bits	Default	Access	Description
linShortDelay	$B6_{HEX}$	[7:0]	$4F_{HEX}$	RW	Delay configuration for gating the LIN SHORT signal. This register defines the number of 4MHz clock cycles where TXD is low and RXD is high before the gating logic of the LIN SHORT signal from the LIN PHY is removed. When RXD becomes low or TXD becomes high, the gating logic is reactivated. **Note:** when `linShortDelay` is set to 0, the TXD and RXD levels are ignored and the LIN SHORT signal is not gated.

3.9.4.4. Register “linWuDelay” – Configuration Register for LIN Wakeup Time

Table 3.66 Register `linWuDelay`

Name	Address	Bits	Default	Access	Description
linWuDelay	$B7_{HEX}$	[4:0]	10100_{BIN}	RW	LIN wakeup time. This register defines the number of 125 kHz clock cycles where LIN-RXD must be low before a LIN wakeup conditions is detected. **Important Warning:** Do not set to 0.
Unused		[7:5]	000_{BIN}	RO	Unused; always write as 0

3.10. SBC OTP

There is a 32x8 bit OTP integrated in the SBC that contains the required trimming data as well as the traceability information. The default (erased) state of the OTP cells is 0. Because some of the programmed trim bits are critical for operation, such as the voltage trim bits, redundancy is implemented for the lower quarter of the OTP memory. This part of the OTP contains only up to four bits of information that are programmed to bits [3:0] as well as to bits [7:4]. During the download procedure, the correct content is determined by combining bit 0 and bit 4, bit 1 and bit 5, bit 2 and bit 6, and bit 3 and bit 7 via an OR gate.

Table 3.67 OTP Memory Map

Name	OTP Address	SPI Address	Size	Copy to Reg.	Redun-dancy	Byte Order	Description
OTP_VALID	00_{HEX}	$E0_{HEX}$	0	No	Yes	---	[0]: OTP content valid
LIN_TRIM	01_{HEX}	$E1_{HEX}$	3:0	Yes	Yes	---	[3:0]: IBIAS_LIN_TRIM[3:0]
VDD_TRIM	02_{HEX}	$E2_{HEX}$	3:0	Yes	Yes	---	[0]: VDDC trim bit [1]: VDDP trim bit [3:2]: vbgh_trim[1:0]
BG_TRIM	03_{HEX}	$E3_{HEX}$	3:0	Yes	Yes	---	[3:0]: vbgh_trim[5:2]
IREF_OSC_0	04_{HEX}	$E4_{HEX}$	3:0	Yes	Yes	LSB	[3:0]: IREF_OSC_TC_TRIM[3:0]
IREF_OSC_1	05_{HEX}	$E5_{HEX}$	3:0	Yes	Yes	MSB LSB	[0]: IREF_OSC_TC_TRIM[4] [2]: IBIAS_LIN_TRIM[4] [3]: IREF_OSC_TRIM[0]
IREF_OSC_2	06_{HEX}	$E6_{HEX}$	3:0	Yes	Yes	---	[3:0]: IREF_OSC_TRIM[4:1]
IREF_OSC_3	07_{HEX}	$E7_{HEX}$	3:0	Yes	Yes	MSB	[3:0]: IREF_OSC_TRIM[8:5]
IREF_LP_OSC	08_{HEX}	$E8_{HEX}$	6:0	Yes	No	---	Trim value for the low-power oscillator
ADCCGAN_0	09_{HEX}	$E9_{HEX}$	7:0	Yes	No	LSB	Gain for the current measurement
ADCCGAN_1	$0A_{HEX}$	EA_{HEX}	7:0	Yes	No	---	
ADCCGAN_2	$0B_{HEX}$	EB_{HEX}	7:0	Yes	No	MSB	
ADCCOFF_0	$0C_{HEX}$	EC_{HEX}	7:0	Yes	No	LSB	Offset for the current measurement
ADCCOFF_1	$0D_{HEX}$	ED_{HEX}	7:0	Yes	No	---	
ADCCOFF_2	$0E_{HEX}$	EE_{HEX}	7:0	Yes	No	MSB	
ADCVGAN_0	$0F_{HEX}$	EF_{HEX}	7:0	Yes	No	LSB	Gain for the voltage measurement
ADCVGAN_1	10_{HEX}	$F0_{HEX}$	7:0	Yes	No	---	
ADCVGAN_2	11_{HEX}	$F1_{HEX}$	7:0	Yes	No	MSB	
ADCVOFF_0	12_{HEX}	$F2_{HEX}$	7:0	Yes	No	LSB	Offset for the voltage measurement
ADCVOFF_1	13_{HEX}	$F3_{HEX}$	7:0	Yes	No	---	
ADCVOFF_2	14_{HEX}	$F4_{HEX}$	7:0	Yes	No	MSB	
ADCTGAN_0	15_{HEX}	$F5_{HEX}$	7:0	Yes	No	LSB	Gain for the temperature measurement
ADCTGAN_1	16_{HEX}	$F6_{HEX}$	7:0	Yes	No	MSB	
ADCTOFF_0	17_{HEX}	$F7_{HEX}$	7:0	Yes	No	LSB	Offset for the temperature measurement
ADCTOFF_1	18_{HEX}	$F8_{HEX}$	7:0	Yes	No	MSB	
---	19_{HEX}	$F9_{HEX}$	---	No	No	---	Unused
LOT_ID_0	$1A_{HEX}$	FA_{HEX}	7:0	No	No	LSB	Lot ID number
LOT_ID_1	$1B_{HEX}$	FB_{HEX}	7:0	No	No	MSB	
WAFER_NO_0	$1C_{HEX}$	FC_{HEX}	7:0	No	No	LSB	Wafer number
WAFER_NO_1	$1D_{HEX}$	FD_{HEX}	7:0	No	No	MSB	
DIE_POS_0	$1E_{HEX}$	FE_{HEX}	7:0	No	No	LSB	Die position
DIE_POS_1	$1F_{HEX}$	FF_{HEX}	7:0	No	No	MSB	

After reset of the SBC, the OTP download procedure is automatically triggered. First, the OTP content is checked for validity (“bit 0 OR bit 4” must be equal to 1). If the content is not valid, the download procedure is stopped. Otherwise, the information stored at OTP addresses 01_{HEX} to 18_{HEX} is copied into the corresponding registers. The download procedure can also be started by the user by writing the value 1 into the `otpDownload` bit in register `cmdExe` (see Table 3.7). Special care must be taken after starting the OTP download procedure as the system must not go to the power-down state as long as the download procedure is active. The status of the download procedure is signaled to the user via the SSW bit 0.

In addition to being read by triggering the OTP download procedure that copies the OTP content into the corresponding registers, the raw contents of the OTP can be read by the user via the SPI interface at SPI addresses $E0_{HEX}$ to FF_{HEX}. This might be useful for checking the content of the OTP. For the lowest quarter of the OTP, this is useful for checking that no bit has changed its value. The user might also choose to implement redundancy for the other values by mirroring the contents into the flash memory on the MCU.

3.11. Miscellaneous Registers

3.11.1.1. Register “pullResEna” – Pull-down Resistor Control Register

Each of the eight internal nodes CSN, SCLK, MOSI, TXD, TRSTN, TCK, TMS, and DBGEN contains a controllable internal pull-down resistor that is active by default. The pull-down resistors are present to prevent a floating input pin if the bonding wire gets broken and to enable the system to detect such a broken wire.

Example: If the bonding wire at TXD is broken, the pull-down resistor would drive TXD low continuously and the LIN TXD timeout detector will trigger and inform the MCU that an error is present.

Directly behind the input pads is a secondary protection stage because VDDP is disabled in some power-down states. The three SPI inputs to the SBC, which are CSN, SCLK and MOSI, as well as the TXD input, are only enabled in full-power (FP) state when the MCU is powered and clocked. The DBGEN input is enabled as long as MCU_RSTN is high (in the FP or LP state) while the three test inputs to the SBC, which are TRSTN, TCK and TMS, are only enabled when TEST is high.

Note: Because the TEST input pad also contains a pull-down resistor, disabling the pull-down resistors for the three test input pins is always safe as long as TEST is low.

© 2016 Integrated Device Technology, Inc.

Table 3.68 Register `pullResEna`

Name	Address	Bits	Default	Access	Description
pullResEnaCsn	$B8_{HEX}$	[0]	1_{BIN}	RW	When set to 1, the pull-down resistor in the CSN pad is connected to the pad.
pullResEnaSpiClk		[1]	1_{BIN}	RW	When set to 1, the pull-down resistor in the SCLK pad is connected to the pad.
pullResEnaMosi		[2]	1_{BIN}	RW	When set to 1, the pull-down resistor in the MOSI pad is connected to the pad.
pullResEnaTxd		[3]	1_{BIN}	RW	When set to 1, the pull-down resistor in the TXD pad is connected to the pad.
pullResEnaTrstn		[4]	1_{BIN}	RW	When set to 1, the pull-down resistor in the TRSTN pad is connected to the pad.
pullResEnaTck		[5]	1_{BIN}	RW	When set to 1, the pull-down resistor in the TCK pad is connected to the pad.
pullResEnaTms		[6]	1_{BIN}	RW	When set to 1, the pull-down resistor in the TMS pad is connected to the pad.
pullResEnaDbgen		[7]	1_{BIN}	RW	When set to 1, the pull-down resistor in the DBGEN pad is connected to the pad.

3.11.1.2. Register “versionCode” – Version Code of SBC

The version code of the SBC is 300_{HEX}.

Table 3.69 Register `versionCode`

Name	Address	Bits	Default	Access	Description
versionCode[7:0]	BA_{HEX}	[7:0]	00_{HEX}	RO	Version code of the SBC.
versionCode[11:8]	BB_{HEX}	[3:0]	0011_{BIN}	RO	
Unused		[7:4]	0000_{BIN}	RO	Unused; always write as 0.

3.11.1.3. Register “pwrTrim” – Trim Register for the Voltage Regulators and Bandgap

Table 3.70 Register `pwrTrim`

Name	Address	Bits	Default	Access	Description
vddcTrim	$C0_{HEX}$	[0]	0_{BIN}	RW	Trim register for VDDC regulator: 0 – VDDC is trimmed to 1.2V 1 – VDDC is trimmed to 1.8V **Note:** This register is set by the OTP download procedure when the OTP content is valid.
vddpTrim		[1]	0_{BIN}	RW	Trim register for the VDDP regulator: 0 – VDDP is trimmed to 2.5V 1 – VDDP is trimmed to 3.3V **Note:** This **register** is set by the OTP download procedure when the OTP content is valid.
vbghTrim		[7:2]	011111_{BIN}	RW	Trim register for the high-precision bandgap. **Note:** This register is set by the OTP download procedure when OTP content is valid.

Important Warning: Changing the settings of bits `vddcTrim` and `vddpTrim` can cause damage to the MCU or cause malfunction of the MCU.

3.11.1.4. Register “ibiasLinTrim” – Trim Register for the Bias Current of the LIN Block

Table 3.71 Register `ibiasLinTrim`

Name	Address	Bits	Default	Access	Description
ibiasLinTrim	$C3_{HEX}$	[4:0]	10000_{BIN}	RW	Trim register for the bias current of the LIN block: 0 – Smallest value 1 – Largest value **Note:** This register is set by the OTP download procedure when OTP contents are valid.
Unused		[7:5]	000_{BIN}	RO	Unused; always write as 0.

3.12. Voltage Regulators

In addition to the battery voltage (VBAT), four additional voltage domains are implemented in the ZSSC1956: the analog voltage VDDA, the digital voltage for low-power mode (VDDL) for the SBC and the RAM of the MCU, the supply voltage (VDDP) for the I/Os of the SBC and the MCU, and the supply voltage (VDDC) for the core of the MCU. The regulators are low-dropout regulators (LDO). The VDDL regulator, which is active in the low-power states, has very low power consumption.

3.12.1. VDDE

The following blocks are connected directly to the VDDE power supply:

- Low-power bandgap
- High-precision bandgap
- High-precision oscillator
- POR
- Regulator for VDDA
- Regulator for VDDL
- Regulator for VDDC
- Regulator for VDDP

3.12.2. VDDA

The analog regulator provides a 2.5V output and can drive up to 10mA of load current. The output voltage is continuously regulated with respect to the bandgap voltage (vbgh). A resistor chain generates the appropriate voltage for the feedback comparison with the bandgap voltage so that the correct voltage is generated. This internal regulated voltage serves as a supply voltage for the analog blocks. The analog regulator can be switched off (e.g., in Sleep Mode).

The following blocks are connected directly to the VDDA analog power supply:

- Level-Shifter
- PGA
- Divider
- Temperature Measurement
- SD-ADC Channel 1 (current)
- SD-ADC Channel 2 (voltage and temperature)
- All blocks necessary for data acquisition (current, voltage and temperature)

© 2016 Integrated Device Technology, Inc.

3.12.3. VDDL

The VDDL regulator provides the necessary supply voltage for the low-power domain and the digital core (1.8V, typical). It is permanently connected to the external supply rail and is always switched on until there is enough voltage on it. The load capabilities depend on the value of the supply voltage as follows:

Table 3.72 VDDL Regulator Load Capabilities

Supply Voltage VDDE	Load Current Capability
$V_{DDE} \geq 3.5V$	Load current ≤ 6mA
$2.0V \leq V_{DDE} < 3.5V$	Load current ≤ 100µA (guaranteed nominal output voltage)
$1.65V \leq V_{DDE} < 2.0V$	Load current ≤ 100µA (guaranteed output voltage above 1.6V)

The regulator supports an IDDQ Test Mode during which its output is fully isolated from the rest of the circuit. Another supported mode is the Low-Power Mode during which the current consumption of the regulator is reduced. Additional circuitry is added for over-voltage protection of the output.

The following blocks are connected directly to VDDL:

- LIN 2.1
- Power Management Unit (supply for flash memory/microcontroller)
- Control Register for the ADCs
- Watchdog

3.12.4. VDDP

The peripheral regulator provides 3.3V. The VDDP regulator can drive up to 30mA of load current and can be switched off (e.g., in Sleep Mode).

The I/O blocks of the SBC and the MCU are connected directly to VDDP.

3.12.5. VDDC

The core regulator provides 1.8V. This regulator can drive up to 40mA of load current and can be switched off (e.g., in Sleep Mode).

The MCU core block is connected directly to VDDC.

4 Functional Block Descriptions for the MCU

4.1. Introduction

The MCU contains an ARM® subsystem, memories (FLASH, RAM, and ROM), and several peripherals for external communication as well as an interface to the SBC. A simplified block diagram is shown in section 2.3, which illustrates the included blocks.

The ARM® subsystem is comprised of the following blocks:

- Cortex™-M0* processor
- Debug controller including
  - JTAG interface
  - 4 hardware break-point comparators
  - 2 hardware watch-point comparators
- Interrupt controller (NVIC) providing the non-maskable interrupt (NMI) and 9 interrupt lines:
  - Interrupt line 0: flash controller interrupt — level-sensitive
  - Interrupt line 1: external interrupt (from SBC) — level-sensitive
  - Interrupt line 2: ahbLIN interrupt — level-sensitive
  - Interrupt line 3: SPIB8 interrupt — level-sensitive
  - Interrupt line 4: 32-bit timer interrupt — pulse
  - Interrupt line 5: GPIO interrupt — level-sensitive
  - Interrupt line 6: SPI2 interrupt (from ZSYSTEM2) — level-sensitive
  - Interrupt line 7: USART interrupt (from ZSYSTEM2) — level-sensitive
  - Interrupt line 8: I²C™ interrupt (from ZSYSTEM2) — level-sensitive
- System timer (SysTick)
- Fast single-cycle multiplier

For details about usage of the ARM® subsystem, see the *IDT ARM® Cortex™ M0 User Guide* and refer to technical documents available from ARM, Ltd.

4.2. Memory Structure

There are two different kinds of physical memory integrated in the MCU: a 96kB flash and an 8kB SRAM block. The peripherals and the required ROM table are also mapped into the memory space. A default slave replies with an error response when unused parts of the memory space are accessed. By default, the flash is mirrored to address 0000 0000_{HEX}. Software can change this and mirror the SRAM to address 0000 0000_{HEX} instead by writing to the `memSwap` field in register SYS_MEMPORTCFG (see Table 4.6).

* ARM® is a registered trademark of ARM Limited. Cortex™ is a trademark of ARM Limited.

4.2.1. Memory Map

Table 4.1 Address Map of MCU

Address	Description
FFFF FFFF$_{HEX}$	Reserved
F000 1000$_{HEX}$	
F000 0FFF$_{HEX}$	System ROM Table
F000 0000$_{HEX}$	
EFFF FFFF$_{HEX}$	Reserved
E010 0000$_{HEX}$	
E00F FFFF$_{HEX}$	Private Peripheral Bus; see ARM® documentation for details
E000 0000$_{HEX}$	
DFFF FFFF$_{HEX}$	Reserved
6000 0000$_{HEX}$	
5FFF FFFF$_{HEX}$	Reserved
4000 2400$_{HEX}$	
4000 23FF$_{HEX}$	SPIB8
4000 2000$_{HEX}$	
4000 1FFF$_{HEX}$	ZSYSTEM2 (SPI2, USART, I²C™†)
4000 1C00$_{HEX}$	
4000 1BFF$_{HEX}$	ahbLIN
4000 1800$_{HEX}$	
4000 17FF$_{HEX}$	GPIO
4000 1400$_{HEX}$	
4000 13FF$_{HEX}$	32-Bit Timer
4000 1000$_{HEX}$	
4000 0FFF$_{HEX}$	Flash Info Page
4000 0C00$_{HEX}$	
4000 0BFF$_{HEX}$	Flash Controller
4000 0800$_{HEX}$	
4000 07FF$_{HEX}$	Reserved
4000 0400$_{HEX}$	
4000 03FF$_{HEX}$	System Management Unit
4000 0000$_{HEX}$	
3FFF FFFF$_{HEX}$	Reserved
2000 2000$_{HEX}$	
2000 1FFF$_{HEX}$	SRAM
2000 0000$_{HEX}$	
1FFF FFFF$_{HEX}$	Reserved
1001 8000$_{HEX}$	
1001 7FFF$_{HEX}$	Flash
1000 0000$_{HEX}$	
0FFF FFFF$_{HEX}$	Reserved
0001 8000$_{HEX}$	
0001 7FFF$_{HEX}$	Depending on the `memSwap` bit, either flash memory (0) or SRAM (1) is mapped to address 0000 0000$_{HEX}$ and higher.
0000 0000$_{HEX}$	

† I²C™ is a trademark of NXP.

4.2.2. Flash Memory

There is a 96kB flash memory integrated into the system to store the boot loader, the program code, and logging data. As the flash memory has some dedicated timings for the control signals when erasing (part of) the flash and when writing data to the flash, a flash controller is used for flash integration into the system to support all mandatory operations (read, write, erase) to be performed on the different flash locations and to guarantee the correct timings for write and erase operations. It is also checked whether the different operations are allowed to be performed depending on the memory protection scheme.

The flash memory consists of two sections: the MAIN area and the INFO pages. Together these sections comprise several pages of 512 bytes each. Each page has four rows, and each row contains 32 words. A flash page is the smallest block that can be erased.

The INFO pages have a total size of 1kB while the size of the MAIN area is 96kB. Each word is protected by ECC logic with a hamming distance of 4, which enables the system to correct a single-bit error and to detect two-bit errors within a word. The correct ECC code bits are automatically appended on each write access to the flash. When a single-bit error within a word is detected during a read access, it is automatically corrected.

The occurrence of bit errors is signaled via dedicated status bits in registers FC_STAT_DATA (see Table 4.18) and FC_STAT_PROG (see Table 4.17) in the flash controller. The status bits distinguish between an erased flash word (all1 flag), the detection and correction of a single-bit error (1Err flag), and the detection of more than one bit error (2Err flag), which is uncorrectable.

The two status register sets are distinguished by the type of flash access:

- FC_STAT_PROG status bits are used when errors occur during an instruction fetch.
  - An instruction fetch to an erased memory location (all1 flag set) or to a memory location with more than one error (not correctable!) will assert an NMI as the program is corrupted.
  - The detection and correction of a single-bit error within a word is signaled via the normal interrupt (ARM® interrupt line 0).
- FC_STAT_DATA status bits are used when errors occur during a load operation.
  - Loads from an erased memory location as well as the detection (and correction) of errors within a word are indicated via the normal interrupt (ARM® interrupt line 0).

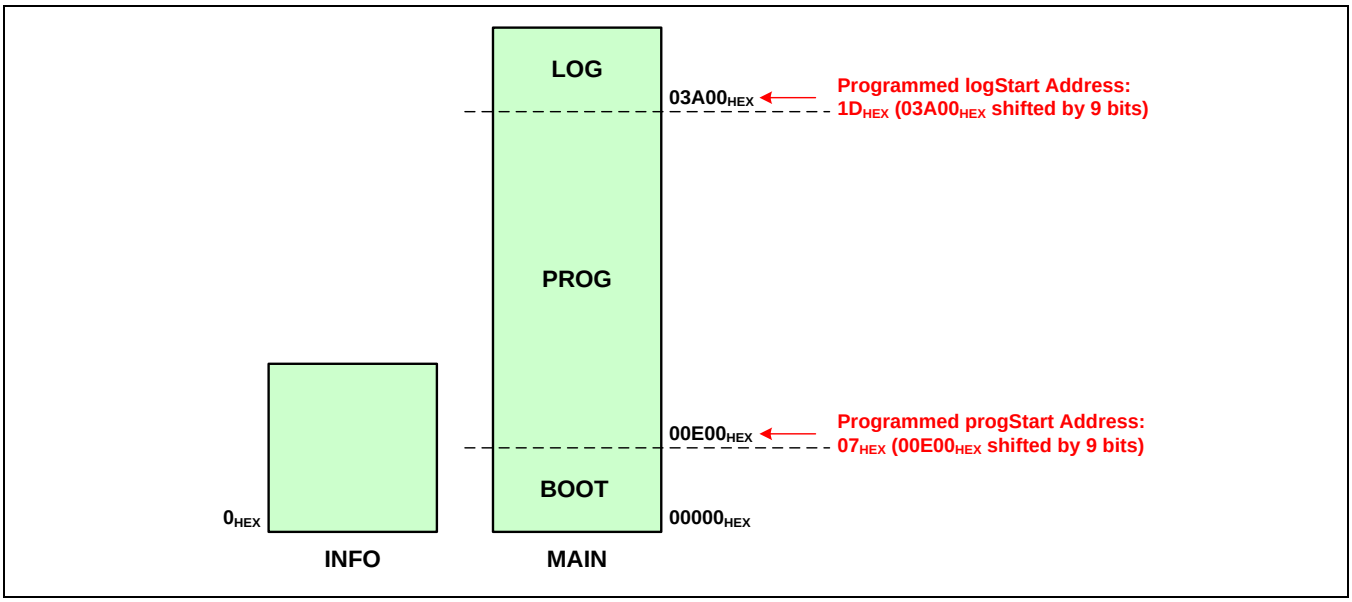
4.2.2.1. MAIN Area

The MAIN area of the flash is physically located at address 1000 0000_{HEX} and higher. It can also be mirrored to address 0000 0000_{HEX} by setting the `memSwap` bit in register SYS_MEMPORTCFG (see Table 4.6) to 0 (default setting). The flash is split into three sections: BOOT, PROG, and LOG (see Figure 4.1). Each section comprises multiple flash blocks of 512 bytes. The BOOT and PROG sections must contain at least one flash block. This partition is only needed for writes when memory protection is active and also for some erase commands.

Note: The flash boundaries can always be extracted from the flash INFO pages or from register SYS_MEMINFO (see Table 4.7). Note that the last nine address bits are not included in the read value as they are always 0. The read value can be interpreted as a flash block number.

The MAIN area can be read with byte, half-word, and word size. It can always be read by the ARM® processor, but reads via the JTAG interface are blocked when memory protection is active. The different sections have no influence on read accesses.

Figure 4.1 Flash Memory Example: BOOT Section of 7 Flash Pages (3.5kB) and PROG Section of 22 Flash Pages (11kB)



As writes must only occur to erased memory locations (all1 flag set when read), one of the four erase commands affecting the MAIN area must be executed before writing to a non-erased memory location. The erase commands are always started via the flash controller.

To prevent an intruder from erasing only a small amount of flash memory and writing a small program in its place that could read out the other memory locations, there are some restrictions for the erase commands:

- ERASE_MAIN command: This command erases the complete MAIN area. Therefore it can always be executed no matter whether memory protection is active or not. It takes approximately 16.3ms. After the execution of this command, writes to all three sections are permitted if memory protection is inactive or as long as the corresponding `allowProg` and `allowBoot` flags in register FC_STAT_CORE are set (see Table 4.16).
- ERASE_PROG command: This command erases the complete PROG section. Therefore it can always be executed no matter whether memory protection is active or not. It is mandatory that the two boundaries are setup correctly. As this command erases all flash pages within the PROG section one after the other, the command execution time is long (approximately 16.3ms per flash page). After the execution of this command, writes to the PROG section are permitted if memory protection is inactive or as long as the `allowProg` flag in register FC_STAT_CORE is set.
- ERASE_BOOT_PROG command: This command erases the complete BOOT and PROG sections. Therefore it can always be executed no matter whether memory protection is active or not. It is mandatory that the upper boundary (PROG to LOG) is setup correctly (see section 4.2.2.2). As this command erases all flash pages within both sections one after the other, the command execution time is long (approximately 16.3ms per flash page). After the execution of this command, writes to all three sections are permitted if memory protection is inactive or as long as the corresponding `allowProg` and `allowBoot` flags in register FC_STAT_CORE are set.

- ERASE_MAIN_PAGE: This command erases a single flash page within the MAIN area, which takes approximately 16.3ms. While this command can always be executed for a flash page in the LOG section, it is rejected for flash pages in the BOOT or PROG section when memory protection is active. It is mandatory that the upper boundary (PROG to LOG) is setup correctly.

Note: As erase commands erase at least a complete flash page, the user must save the data that must be retained from the flash page to be erased.

Note: The allow flags will be cleared by writing 1 to the `clrAllow` bit of register FC_STAT_CORE or by a reset.

The MAIN area can only be written with word size as the ECC code is word-based. Writing one word requires approximately 72µs. There are two ways of writing into the MAIN area: direct write or executing the WRITE command by the flash controller. Direct writes can always be performed by the ARM® processor or via the JTAG interface when the memory protection is inactive. When memory protection is active, no direct write is possible via the JTAG interface. In that case, the ARM® processor can only perform a direct write to the LOG section while direct writes to the BOOT or PROG section are rejected if the corresponding `allowProg` and `allowBoot` flags in register FC_STAT_CORE are not set.

The WRITE command can always be executed by the ARM® processor or via the JTAG interface, but the command is rejected if the write would be performed to the BOOT or PROG section when memory protection is active and the corresponding `allowProg` and `allowBoot` flags in register FC_STAT_CORE are not set.

4.2.2.2. INFO Pages

The INFO pages are mapped into the system address space ranging from 4000 $0C00_{HEX}$ to 4000 $0FFF_{HEX}$. No memory location in the INFO pages can be directly written.

The first page (lower half) contains traceability information as well as a copy of the OTP contents of the SBC. This page is only readable and cannot be accessed indirectly by the flash controller (no write, no erase).

Table 4.2 Memory Content of the Lower INFO Page

INFO Page 0		
Local Address	**Size**	**Contents**
000_{HEX}	1 word	Lot-Wafer-ID
004_{HEX}	1 word	Lot-Wafer-ID
008_{HEX}	1 word	X&Y coordinate
$00C_{HEX}$	1 word	X&Y coordinate
010_{HEX} to $07F_{HEX}$	---	Empty
080_{HEX} to $09F_{HEX}$	8 words	OTP content
$0A0_{HEX}$ to $1EF_{HEX}$	---	Empty
$1F0_{HEX}$ to $1FF_{HEX}$	---	Tester information

The second page (upper half) contains information about memory protection and the three boundaries required to split the flash MAIN area into three sections and the RAM into two sections. The lower half of the second flash page (address offset 200_{HEX} to $2FF_{HEX}$) is always readable while the upper half of the second flash page (address offset 300_{HEX} to $3FF_{HEX}$) can only be read when memory protection is inactive. The different fields can only be modified by different flash controller commands.

When address $20C_{HEX}$ is set to 0000 0000_{HEX} the key-based lock is active, and when address 208_{HEX} is set to 0000 0000_{HEX}, the permanent lock is active. A permanent lock has a higher priority than the key-based lock. The addresses 200_{HEX} to 208_{HEX} are also interpreted as a counter for failed attempts to unlock the key-based lock. Each time an unlock command fails, one of these values is set to 0_{HEX} causing the permanent lock to be activated after the third failure.

The word at address 210_{HEX} contains the three boundaries. The lowest byte of this word contains the flash page number of the first flash page of the PROG section. The second byte of this word contains the flash page number of the first flash page of the LOG section while the highest two bytes of this word contain the RAM word address (without the last two 0's) where the accessible RAM space starts. The complete RAM is always accessible by the ARM® processor no matter whether memory protection is active or not. The lower part of the RAM (below the RAM word address) can only be accessed via the JTAG interface when memory protection is not active. Placing critical elements such as the stack in the RAM part below this RAM word address is recommended to protect the system against intrusion. The upper part of the RAM (starting with the RAM word address) is always accessible via the JTAG interface and can be used for programming the flash using the WRITE command.

Table 4.3 Memory Content of the Upper INFO Page

INFO Page 1		
Local Address	**Size**	**Content**
200_{HEX}	1 word	Count 0
204_{HEX}	1 word	Count 1
208_{HEX}	1 word	Count 2 / permanent lock
$20C_{HEX}$	1 word	Key-based lock
210_{HEX}	1 word	Boundaries for flash and RAM 1 byte (progStart) 1 byte (logStart) 2 bytes (ramSplit)
214_{HEX} to $2FF_{HEX}$	---	Empty
300_{HEX}	1 word	Key length
304_{HEX} to $3??_{HEX}$	N words	Key (maximum 31 words)
$3??_{HEX}$ to $3FF_{HEX}$	---	Empty

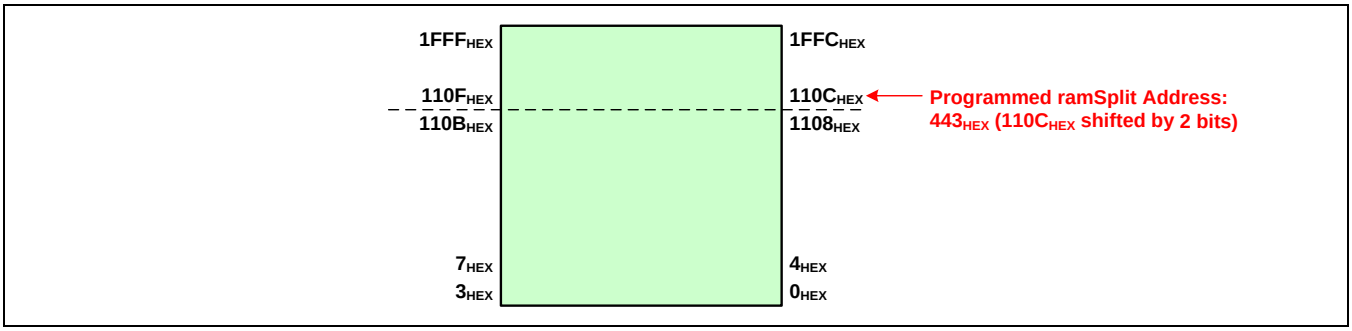
Addresses 300_{HEX} and following (maximum to $37F_{HEX}$) contain the key information for the key-based lock.

4.2.3. RAM Memory

There is 8kB of SRAM memory integrated into the system. The SRAM is physically located at addresses 2000 0000_{HEX} and higher. It can be accessed (read and write) with byte, half-word, and word size. The SRAM can also be mirrored to address 0000 0000_{HEX} by setting the `memSwap` bit in the register SYS_MEMPORTCFG to 1 (see Table 4.6).

The complete RAM is always accessible by the ARM® processor, but there are restrictions for RAM access via the JTAG interface. From the JTAG perspective, the RAM is split into two sections with a configurable boundary (`ramSplit` address). The `ramSplit` address is word-aligned and stored in the flash (see Figure 4.2). The upper section starting with address `ramSplit` can always be accessed via JTAG and can be used for flash reprogramming or to remove the key-based memory protection. The lower section can only be accessed via JTAG when no memory protection is active. The stack used by software will be placed into the lower section to prevent an intruder from corrupting the stack.

Figure 4.2 Example for ramSplit Address



Note: Always place the software stack into the lower section of the RAM. The lower section ends one word address before the address `ramSplit`.

Note: The `ramSplit` address can always be extracted from the flash INFO pages or from register SYS_MEMINFO. Note that the last two address bits are not included in the read value as they are always 0.

4.2.4. System ROM Table

The system ROM table is used to identify the system by an external debugger. The system ROM is mapped into the system address space ranging from F000 0000$_{HEX}$ to F000_0FFF$_{HEX}$. Bit 0 of local address 000$_{HEX}$ can be used by an external debugger to determine whether the memory protection is active (bit is 0) or not as the processor ROM table is not accessible.

Table 4.4 Memory Content of System ROM

Address	Value	Description
FFC$_{HEX}$	0000 00B1$_{HEX}$	[7:0] component ID3 – preamble
FF8$_{HEX}$	0000 0005$_{HEX}$	[7:0] component ID2 – preamble
FF4$_{HEX}$	0000 0010$_{HEX}$	[7:4] component ID1 – component class [3:0] component ID1 – preamble
FF0$_{HEX}$	0000 000D$_{HEX}$	[7:0] component ID0 – preamble
FEC$_{HEX}$	0000 0000$_{HEX}$	[7:4] peripheral ID3 – revision number
FE8$_{HEX}$	0000 009F$_{HEX}$	[7:4] peripheral ID2 – project number [3:0] [3] peripheral ID2 – JEDEC assigned ID fields [2:0] peripheral ID2 – JEP106 ID code [6:4]
FE4$_{HEX}$	0000 0071$_{HEX}$	[7:4] peripheral ID1 – JEP106 ID code [3:0] [3:2] peripheral ID1 – variant [1:0] peripheral ID1 – project number [13:12]
FE0$_{HEX}$	0000 0071$_{HEX}$	[7:0] peripheral ID0 – project number [11:4]
FDC$_{HEX}$	0000 0000$_{HEX}$	[7:0] peripheral ID7 – reserved
FD8$_{HEX}$	0000 0000$_{HEX}$	[7:0] peripheral ID6 – reserved
FD4$_{HEX}$	0000 0000$_{HEX}$	[7:0] peripheral ID5 – reserved
FD0$_{HEX}$	0000 0000$_{HEX}$	[3:0] peripheral ID4 – JEP106 continuation code
FCC$_{HEX}$	0000 0001$_{HEX}$	[0] memType – indicates that the system memory is accessible via the DAP
004$_{HEX}$ to FC8$_{HEX}$	0000 0000$_{HEX}$	Reserved
000$_{HEX}$	0F00 FF0X$_{HEX}$	[31:12] address offset of next ROM table relative to this ROM table [11:2] reserved [1] 32-bit format ROM table → 1 [0] 0: next ROM table is not present; true when memory protection is active 1: next ROM table is present; true when memory protection is not active

4.2.5. Memory Protection

Memory protection can be activated via dedicated commands of the flash controller to protect the software stored in the flash. Two different types of memory protection are implemented: a key-based lock and a permanent lock. Both restrict the access possibilities in the same manner, but the key-based lock can be removed by an unlock procedure re-allowing access to the memory again. The user has three attempts to unlock the flash when the key-based lock is active; after the third failed attempt, the flash is locked permanently. It is also possible to unlock the flash from both types of memory protection by erasing the complete PROG section and then erasing the upper INFO page.

When memory protection is active:

- The lower section of RAM cannot be accessed via JTAG interface. The upper section starting at the `ramSplit` address is still accessible via JTAG for reprogramming the flash after erasing it or to unlock the key-based lock.
- The complete flash MAIN area cannot be accessed via JTAG interface.
- Direct writes by the ARM® processor to the BOOT and PROG sections are rejected when corresponding allow flags are not set (by a previous erase command).
- All JTAG accesses to the PPB area (system address space from E000 0000_{HEX} to EFFF $FFFF_{HEX}$) except to the DHCSR (Debug Halting Control and Status Register) at address E000 $EDF0_{HEX}$) (refer to the *IDT ARM® Cortex™ M0 User Guide*) are rejected.
- For JTAG writes to DHCSR, data bit 2 (STEP) is forced to 0 to avoid debug stepping.
- The flash controller commands that are allowed to be executed are restricted.

The type of memory protection as well as the number of failed unlock attempts can be read from register SYS_MEMINFO in the system management unit (SMU), which is always the reference as there might be an inconsistency between this register and the contents of the flash INFO page after successful execution of an UNLOCK_CMD (see Table 4.9).

4.3. System Management Unit

The system management unit (SMU) generates all internal clocks and resets. It also provides some support for the different power-down modes controlled by the SBC and contains some registers for system configuration.

4.3.1. Resets

There are five different reset sources in the system:

- External reset provided via the MCU_RSTN internal node generated by the SBC.
- External JTAG reset provided via the TRSTN pin.
- System reset request generated in the ARM® core by writing to register AIRCR (refer to the *IDT ARM® Cortex™ M0 User Guide*). This register is always accessible by the ARM® processor itself, but it can only be accessed via JTAG if the memory protection is not active.
- JTAG reset request generated in the SMU by writing to register SYS_RSTSTAT (see Table 4.8) in the SMU. This reset can always be generated via the JTAG interface. It cannot be generated by the ARM® processor itself.
- System lockup. This reset can only occur when the ARM® processor detects a lockup and has previously enabled this reset source (disabled by default) by writing to register SYS_RSTSTAT.

The external reset via the MCU_RSTN pad resets all logic except the JTAG state machine, which is reset by the external JTAG reset only.

A reset caused by one of the other three reset sources sets the ARM® core back into its default state except the debug logic. It also resets all peripherals except the following registers in the SMU:

- Register SYS_MEMINFO (see Table 4.7)
- The memSwap bit of register SYS_MEMPORTCFG (see Table 4.6)
- The four reset status bits within register SYS_RSTSTAT (see Table 4.8)

4.3.2. Clocks

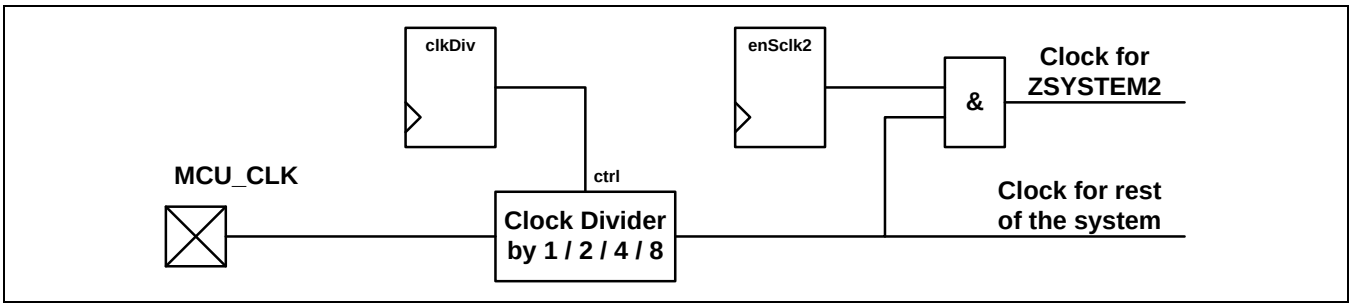
The system has two different clock sources:

- Internal clock provided via the MCU_CLK pad generated by the SBC (typically 20MHz)
- External JTAG clock provided via the TCK pin

The TCK clock is only used for the JTAG access point. The rest of the MCU is clocked by a divided clock derived from MCU_CLK. By default, MCU_CLK is divided by 1, but a divide value of 2, 4, and 8 can be configured via the bit field clkDiv in register SYS_CLKCFG (see Table 4.5) causing the system to run at 10, 5, or 2.5 MHz respectively. While most of the system is clocked during normal operating mode, the clock for ZSYSTEM2 is disabled by default and must be enabled before any access to ZSYSTEM2 takes place. To enable the clock for ZSYSTEM2, set the enSclk2 bit to 1 in register SYS_CLKCFG (see Table 4.5).

Important: Do not access ZSYSTEM2 when its clock is disabled.

Figure 4.3 System Clocks



Important Warning: Special care must be taken if changing the clock divider as the operation of several parts of the system depends on the clock frequency.

Do *not* change the clkDiv value when any of these conditions exist:

- Any flash command is executed by the flash controller. Otherwise the flash can be destroyed. Wait until command execution has finished.
- ahbLIN is active as the inactivity timer, and the Break Sync Detector depends on the clock frequency. When ahbLIN is transmitting, wait until the transmission has completed. The ahbLIN can be placed into a safe state for changing the clock divider value by changing these bits in the Z1_LINCFG register: write 0 to the enToCnt field and write 1 to the stopRx and disableRx fields (see Table 4.32).
- Any operation relying on the 10ms reference of the SYST_CALIB register of the ARM® core is active (see the *IDT ARM® Cortex™ M0 User Guide*). These operations must be stopped first.

- The USART or I²C™ module in ZSYSTEM2 is enabled as their baud rates depend on the clock frequency. These modules must be disabled first.
- Any slave connected to the SPI in ZSYSTEM2 requires a constant SPI frequency. In this case, an active SPI transfer must finish first. The SBC does not need a constant frequency on its SPI. Therefore the SPIB8 must only be stopped when changing the clock divider value would cause a SPI frequency higher than 5MHz.

4.3.3. Power Modes

There are three power modes within the MCU: the Normal Mode equal to the MCU-ON state on the system level (see section 2.4), the Sleep Mode equal to the MCU-SLP state on the system level, and the DeepSleep Mode. In Normal Mode, the ARM® core and the peripherals are clocked and software is executed. This state is entered after power-up or when the system wakes up. The other two modes are distinguished by an ARM® internal register, the system control register (SCR), and its SLEEPDEEP bit, which is controlled by software. Both sleep modes are entered by the WFI (wait for interrupt) instruction.

If the SLEEPDEEP bit is not set on executing of a WFI instruction, the Sleep Mode is entered. In Sleep Mode, the clock for the ARM® core is stopped, but the peripherals remain clocked to be able to wake up the ARM® core if an interrupt occurs and this interrupt is enabled. If an enabled interrupt occurs, the clock for the ARM® core is re-enabled and the system continues in Normal Mode at the position where it was stopped.

Important Warning: Do not enter Sleep Mode after sending a “power-down” command to the SBC. When the SBC enters one of its power-down states (LP, ULP or OFF), at least the MCU_CLK is stopped on SBC side. This can cause peripherals to stop in an unsafe state, which can cause damage to the system. The Sleep Mode must only be entered when the SBC remains in its FP state.

Note: When an interrupt source is disabled, it cannot wake up the system.

If the SLEEPDEEP bit is set on executing of a WFI instruction, the DeepSleep Mode is entered on the MCU. When the WFI instruction is executed without sending a power-down command to the SBC in advance, the SBC remains in its FP state. In this case, the MCU_CLK is only gated at the MCU input while the SBC continues its generation. This combination of power states (SBC in FP state, MCU in DeepSleep Mode) is called the MCU-DEEP state on the system level. In the MCU-DEEP state, the MCU can only wake up by an active SBC interrupt. It is mandatory that a source in the SBC is enabled to generate the interrupt and that the SBC interrupt is enabled in the MCU.

When the WFI instruction is executed after sending a power-down command to the SBC, the CSN line of the SPI connecting to the SBC will be released safely by the WFI instruction. This release of CSN triggers the SBC to enter the configured power-down state. The CSN line must not be directly released by software as the MCU might be in an intermediate state when the MCU_CLK clock and/or power are switched off on the SBC side.

© 2016 Integrated Device Technology, Inc.

There are three wake-up scenarios from the MCU's DeepSleep Mode after a power-down command has been sent to the SBC:

1. When SBC enters its ULP or OFF state (same power-down state on the system level), it stops the generation of the MCU_CLK clock and disables the power for the pads and MCU core (VDDP and VDDC); however, the RAM remains powered (VDDL). To protect the RAM contents from being corrupted during that time, the MCU input RAM_PROTN is driven low by the SBC, which places all RAM inputs into a safe state. When the MCU wakes up, the SBC re-enables all power supplies and the MCU clock and generates a reset via the MCU_RSTN pad. After the reset is released, the MCU starts again as after power-up.
2. When the SBC enters its LP state, it only stops the generation of the clock MCU_CLK but the MCU remains fully powered. When the MCU wakes up, the SBC re-enables the clock and asserts the interrupt line IRQN. Because the MCU is not reset, it restarts where it was stopped or enters the interrupt service routine. It is important that the ARM® core has enabled the SBC interrupt for correct wakeup as it has stopped its internal clock when entering its DeepSleep Mode.
3. When the SBC receives a power-down command when it has already detected an interrupt, it does not enter the desired power-down state. It stays in its FP state but asserts the interrupt line IRQN. It is important that the MCU has enabled the SBC interrupt for the correct wakeup as it has stopped its internal clock when entering its DeepSleep Mode.

Important Warning: Always enable the SBC interrupt in the NVIC before going to Sleep Mode or DeepSleep Mode as the SBC rejects the power-down commands when it has an active interrupt. Otherwise the system can hang up.

Important Warning: Do not enter DeepSleep Mode when any flash command is active. Otherwise the flash could be destroyed.

Important: Stop the ahbLIN, USART and I²C™ to prevent incorrect behavior after wakeup from the LP state or when the SBC rejects the power-down command due to an active interrupt.

4.3.4. Pin Configuration

While all the pins from/to the SBC have a fixed behavior, the functionality of the GPIO pads can be changed by software.

Recommendation: The MCU has 16 GPIOs, but only 5 of them are bonded: GPIO0, GPIO1, GPIO2, GPIO3, GPIO4. Keeping the unbonded GPIO pads switched as inputs is recommended.

By default, all GPIO pads operate as pure GPIOs switched as inputs and configured with open-drain output driver functionality. The GPIO pins require an external pull-up resistor if they will operate as open-drain outputs.

© 2016 Integrated Device Technology, Inc.

The following bit fields in register SYS_MEMPORTCFG (see Table 4.6) can be used to change the behavior and functionality of the GPIO pads:

- `ppNod`: This 16-bit register field is used to individually select the output driver behavior of each GPIO pad. Each GPIO can be configured to operate as push-pull or open-drain (default). This configured output driver behavior is also used when one of the peripheral modules of ZSYSTEM2 is mapped on the corresponding GPIO pad with the following exceptions:
  - The SCL line of I²C™ always operates as open-drain independent of the `ppNod` setting.
  - The SDA line of I²C™ always operates as open-drain independent of the `ppNod` setting.
  - The RXD line of USART always operates as open-drain independent of the `ppNod` setting.
- `spiCfg`: This 2-bit register field is used to connect the SPI of ZSYSTEM2 to the GPIO pads.
  - The lower bit of this 2-bit register field is used to enable the connections between the GPIO pads and the SPI of ZSYSTEM2. When enabled, the GPIO functionality is not available.
  - The upper bit is used to distinguish between two GPIO pad sets to which the SPI lines are connected when the connections are enabled by the lower bit:
    - Mapping when the upper bit is 0:
      - CSN GPIO[0]
      - MOSI GPIO[1]
      - MISO GPIO[2]
      - SCLK GPIO[3]
    - Mapping when upper bit is 1 (do not use this setting as these GPIO pads are not bonded):
      - CSN GPIO[9]
      - MOSI GPIO[10]
      - MISO GPIO[11]
      - SCLK GPIO[12]
  - The output behavior of the three SPI output lines (SCLK, CSN, MOSI) are determined by the settings of the `ppNod` field.
- `usartCfg`: This 3-bit register field is used to connect the USART of ZSYSTEM2 to the GPIO pads.
  - The lowest bit of this 3-bit register field is used to enable the connections between the GPIO pads and the USART of ZSYSTEM2. When enabled, the GPIO functionality is not available.
  - The upper two bits are used to select between four GPIO pad sets to which the USART pins are connected when the connections are enabled by the lowest bit in `usartCfg`:
    - Mapping when the upper two bits are 00_{BIN} (only the TXD line is usable as the GPIO[8] pad is not bonded):
      - TXD GPIO[4]
      - RXD GPIO[8]
    - Mapping when the upper two bits are 01_{BIN}:
      - TXD GPIO[2]
      - RXD GPIO[3]

        - Mapping when the upper two bits are 10_{BIN} (do not use this setting as these GPIO pads are not bonded):
            - TXD GPIO[6]
            - RXD GPIO[7]
        - Mapping when the upper two bits are 11_{BIN} (do not use this setting as these GPIO pads are not bonded):
            - TXD GPIO[9]
            - RXD GPIO[10]
    - The output behavior of the TXD line is determined by the settings of the `ppNod` field. The RXD line always operates as open-drain (`ppNod` setting is overridden).
- `i2cCfg`: This 3-bit register field is used to connect the I²C™ of ZSYSTEM2 to the GPIO pads.
    - The lowest bit of this 3-bit register field is used to enable the connections between the GPIO pads and the I²C™ of ZSYSTEM2. When enabled, the GPIO functionality is not available.
    - The upper two bits are used to select between four GPIO pad sets to which the I²C™ lines are connected when the connections are enabled by the lowest bit in `i2cCfg`:
        - Mapping when the upper two bits are 00_{BIN} (do not use this setting as these GPIO pads are not bonded):
            - SCL GPIO[11]
            - SDA GPIO[12]
        - Mapping when the upper two bits are 01_{BIN}:
            - SCL GPIO[0]
            - SDA GPIO[1]
        - Mapping when the upper two bits are 10_{BIN} (do not use this setting as the GPIO[5] pad is not bonded):
            - SCL GPIO[4]
            - SDA GPIO[5]
        - Mapping when the upper two bits are 11_{BIN} (do not use this setting as these GPIO pads are not bonded):
            - SCL GPIO[14]
            - SDA GPIO[15]
    - Both I²C™ lines (SCL, SDA) always operate as open-drain (`ppNod` setting is overridden).
- `linTest:` This 1-bit register field is used to provide a direct connection between the GPIO pads and the TXD and RXD pads connected to the SBC. This must be used for the LIN conformance test where direct access to the LIN-PHY is needed.
    - TXD line is connected to GPIO[4] pin for direct control.
    - RXD line is connected to GPIO[2] pin for direct observation.

Note: GPIO functionality is only available when no other interface is mapped onto the specific GPIO pad.

Note: As can be seen from the description above, different interface modules can be mapped onto the same GPIO pads. When more than one interface is mapped onto a GPIO pad, only one functionality is enabled with the following priority (highest priority first):

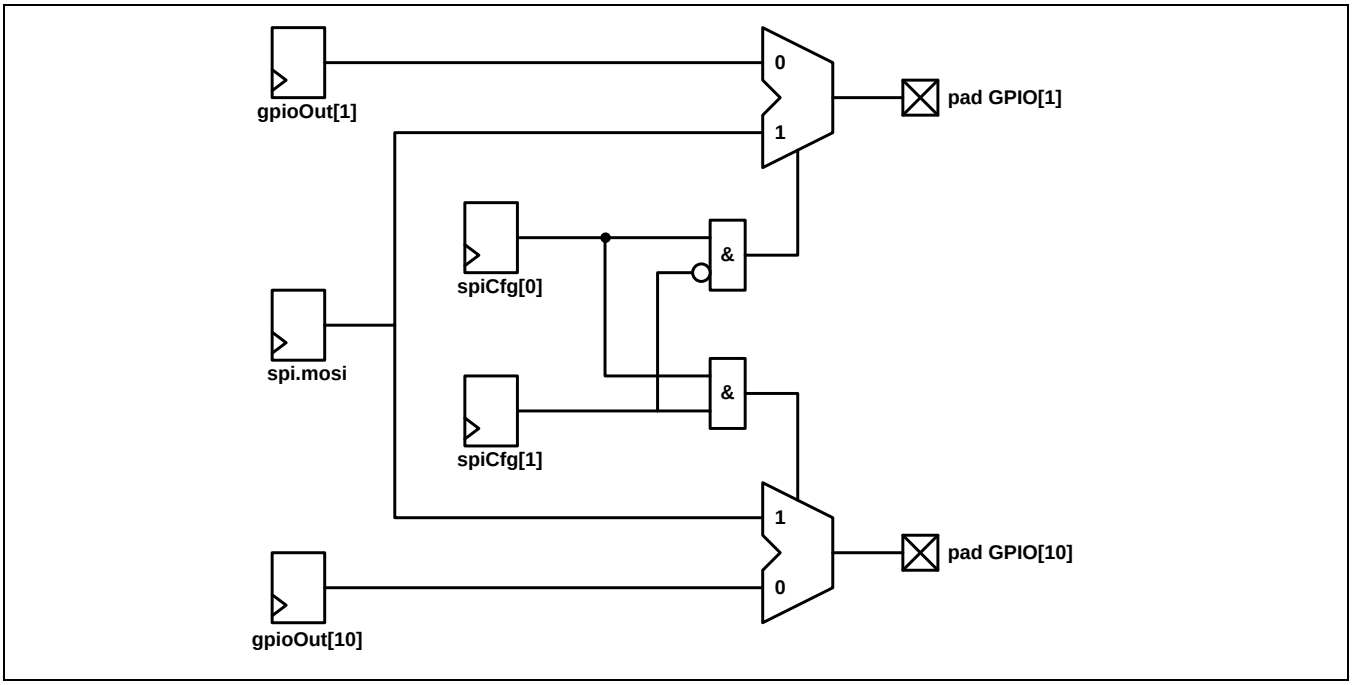
- linTest
- I^2C™
- USART
- SPI
- GPIO

Note: For software debugging purposes when only an output for printing debug information is needed, the USART variant 0 (TXD at GPIO[4]; RXD at GPIO[8]) can be used although GPIO[8] is not bonded.

Example: As can be seen from the above mapping, the MOSI line of the SPI within ZSYSTEM2 can be mapped to pad GPIO[1] or to the unbonded pad GPIO[10]. For simplification, the other mapping possibilities are not considered for this example. To enable the connection for the SPI, the `spiCfg`[0] bit in register SYS_MEMPORTCFG must be set to 1; otherwise both GPIO pads would be driven by the corresponding registers from the GPIO module. `spiCfg`[1] is used to select the GPIO pad where MOSI will be connected. The other GPIO pad remains connected to the GPIO module.

Figure 4.4 Example for Mapping MOSI of the SPI in ZSYSTEM2 to the GPIO Pads

Bit `spiCfg[0]` enables any connection; bit `spiCfg[1]` selects the appropriate GPIO pad.



4.3.5. SMU Module Register Overview

4.3.5.1. Register "SYS_CLKCFG" – Clock Configuration

Table 4.5 Register SYS_CLKCFG – system address 4000 0000$_{HEX}$

Name	Bits	Default	Access	Description
clkDiv	[1:0]	00_{BIN}	RW	Clock divider value 0: incoming clock is divided by 1 1: incoming clock is divided by 2 2: incoming clock is divided by 4 3: incoming clock is divided by 8
Unused	[6:2]	00000_{BIN}	RO	Unused; always write as 0.
enSclk2	[7]	0_{BIN}	RW	Enable bit for ZSYSTEM2 clock
Unused	[31:8]	00 0000_{HEX}	RO	Unused; always write as 0.

4.3.5.2. Register "SYS_MEMPORTCFG" – Memory and Port Configuration

Table 4.6 Register SYS_MEMPORTCFG – system address 4000 0004$_{HEX}$

Name	Bits	Default	Access	Description
ppNod	[15:0]	0000_{HEX}	RW	Output configuration bits. These 16 bits select individually for each GPIO whether the pad will work as an open-drain (set to 0) or as a push-pull (set to 1).
spiCfg	[17:16]	00_{BIN}	RW	Configuration bits for SPI in ZSYSTEM2. [0]: enable connection between GPIOs and SPI [1]: select 1 of 2 port sets to which SPI will be mapped.
usartCfg	[20:18]	000_{BIN}	RW	Configuration bits for USART in ZSYSTEM2. [0]: enable connection between GPIOs and USART [2:1]: select 1 of 4 port sets to which USART will be mapped. **Important:** The RXD line is always open-drain; settings from ppNod are overridden.
i2cCfg	[23:21]	000_{BIN}	RW	Configuration bits for I²C™ in ZSYSTEM2. [0]: enable connection between the GPIOs and I²C™ [2:1]: select 1 of 4 port sets to which I²C™ will be mapped. **Important:** Both I²C™ lines are always open-drain; settings from ppNod are overridden.
Unused	[29:24]	000000_{BIN}	RO	Unused; always read as 0.
linTest	[30]	0_{BIN}	RW	Configuration bit for LIN test. When set, RXD and TXD line are directly connected to GPIO.
memSwap	[31]	0_{BIN}	RW	Memory swap bit. This bit selects whether the flash MAIN area (set to 0) or the RAM (set to 1) is mirrored to system address 0000 0000_{HEX}.

4.3.5.3. Register "SYS_MEMINFO" – Memory Information

Table 4.7 Register SYS_MEMINFO – system address 4000 0008$_{HEX}$

Name	Bits	Default	Access	Description
progStart	[7:0]	FF_{HEX}	RO	This register represents the first page where the PROG (program) section in the flash memory starts. This value is read from the flash memory during the power-up phase and is updated by some flash commands. **Important:** To get the correct flash address offset, nine 0's must be appended.
logStart	[15:8]	FF_{HEX}	RO	This register represents the first page where the LOG section in the flash memory starts. This value is read from the flash during the power-up phase and is updated by some flash commands. **Important:** to get the correct flash address offset, nine 0's must be appended.
ramSplit	[26:16]	111 1111 1111_{BIN}	RO	This register represents the first RAM word that is accessible by JTAG although the memory is locked. This value is read from the flash during the power-up phase and is updated by some flash commands. **Important:** to get the correct RAM address offset, two 0's must be appended.
Unused	[27]	0_{BIN}	RO	Unused; always read as 0.
protInfo	[31:28]	1111_{BIN}	RO	This register represents the memory protection scheme. This value is read from flash during the power-up phase and is updated by some flash commands. [0]: key based lock [1]: permanent lock [3:2]: number of failed unlock attempts

4.3.5.4. Register "SYS_RSTSTAT" – Reset Status

Table 4.8 Register SYS_RSTSTAT – system address 4000 000C$_{HEX}$

Name	Bits	Default	Access	Description
extRst	[0]	1_{BIN}	RC	This bit indicates if an external reset has occurred. It is cleared when the register is read.
sysRstReq	[1]	0_{BIN}	RC	This bit indicates if a reset forced by a system reset request has occurred. It is cleared when the register is read.
lockupRst	[2]	0_{BIN}	RC	This bit indicates if a reset was forced by a detected lockup when this reset was enabled. It is cleared when the register is read.
jtagRst	[3]	0_{BIN}	RC	This bit indicates if a reset forced by a JTAG reset request has occurred. It is cleared when the register is read.
Unused	[6:4]	000_{BIN}	RO	Unused; always read as 0.
enLockup	[7]	0_{BIN}	RO	This bit indicates whether a lockup from the ARM® core is allowed to reset the system (set to 1) or not (set to 0).
setEnLockup	[7:0]	00_{HEX}	WO	To enable the lockup reset, $C9_{HEX}$ must be written to bits 7:0. It can be disabled by writing a different value. This field is reset by all four reset sources (`extRst`, `sysRstReq`, `lockupRst`, `jtagRst`).

Name	Bits	Default	Access	Description
Unused	[15:8]	00_{HEX}	RO	Unused; always read as 0.
jtagRstReq	[23:16]	00_{HEX}	WO	To generate a reset via the JTAG interface, $3A_{HEX}$ must be written to bits 23:16. These bits cannot be written by the ARM® core.
Unused	[31:24]	00_{HEX}	RO	Unused; always read as 0.

4.4. Flash Controller

The flash controller handles all accesses to the different flash locations (INFO pages; MAIN area). It takes care of the memory protection by rejecting illegal accesses; it provides different types of commands for modifying the flash content; it guarantees all required timings for the different accesses; and it appends and checks the ECC code for robustness of the flash content. It also contains a set of registers that are needed for command execution, which reflect the status of the flash controller and the flash and that enable the different interrupt sources to drive the interrupt line. The interrupt is active-high and is connected to ARM® interrupt 0. Additionally, there are two non-maskable interrupt sources that are connected to the NMI of the ARM® core.

Read accesses: The flash MAIN area containing the BOOT, PROG, and LOG section can always be read by the ARM® processor while it can only be read via JTAG when no memory protection (key-based or permanent lock) is active (register SYS_MEMINFO[29:28] == 0) to prevent unauthorized reading of the program. The lower three quarters of the flash INFO pages and all flash controller registers can always be read by the ARM® processor or via JTAG. The upper quarter of the flash INFO pages containing the key for the key-based lock is only readable when no memory protection is active. All read accesses can be performed with byte, half-word, and word size.

Direct write accesses: The INFO pages cannot be directly written, while the registers are always writable by the ARM® processor and via JTAG. Writes to the registers can be performed with byte, half-word, and word size. When no memory protection is active, the complete MAIN area can be written directly by the ARM® core or via JTAG, but it is the responsibility of the user that only erased memory locations are written. When memory protection is active, only writes to the LOG section performed by the ARM® processor are possible. All other writes to the MAIN area (other section or via JTAG) are rejected. All writes to the MAIN area can only be performed with word size.

Commands: Several commands are implemented for erase and write operations as well as to configure the system (memory boundaries and protection). These commands can always be configured and started by the ARM® processor or via JTAG, but under certain conditions their execution is rejected.

Read status information: There are two registers, FC_STAT_PROG (see Table 4.17) and FC_STAT_DATA (see Table 4.18), for signaling that an error occurred when reading the flash (INFO pages or MAIN area) and for indicating whether the error occurred during an instruction fetch by the ARM® processor (FC_STAT_PROG is used) or during a load operation by the ARM® processor or via JTAG.

© 2016 Integrated Device Technology, Inc.

Three different types of error are signaled:

- The occurrence of a single error within a word that was corrected by the ECC logic is signaled via flags `prog1Err` in FC_STAT_PROG or `data1Err` in FC_STAT_DATA. Both flags are cleared on read access to the corresponding register and can be enabled via register FC_IRQ_EN (see Table 4.15) to drive the normal interrupt line.
- When an empty memory location is read, the flags `progAll1` in FC_STAT_PROG or `dataAll1` in FC_STAT_DATA is set. Both flags are cleared on read access to the corresponding register. While the `dataAll1` flag can be enabled via register FC_IRQ_EN to drive the normal interrupt line (this flag is useful to determine an empty memory location for a write operation to flash), the `progAll1` generates a non-maskable interrupt (NMI) as this situation is a severe problem for software execution.
- The occurrence of more than one error within a word that is not correctable by the ECC logic is signaled via flags `prog2Err` in FC_STAT_PROG or `data2Err` in FC_STAT_DATA. Both flags are cleared on read access to the corresponding register. While the `data2Err` flag can be enabled via register FC_IRQ_EN to drive the normal interrupt line, the `prog2Err` generates a non-maskable interrupt (NMI) as this situation is a severe problem for software execution.

In addition to the error flags, the address of the error position is stored. When consecutive errors occur, the address of the first error is kept until the status register is read. When different types of error occurred (only one of the three flags is set at a time), it cannot be determined from the read status value to which type of error the address belongs. It is also not visible when several errors of the same type occurred.

Command status information: The FC_STAT_CORE register (see Table 4.16) contains eight different status flags about the command execution and the access rights to different memory locations when the protection is active. Four of them are cleared on read access and can be enabled via register FC_IRQ_EN to drive the normal interrupt line. The three “allow” flags are read-only. To clear them for reactivation of the memory protection, a 1 must be written to the bit `clrAllow`. If the “allow” flags are not cleared, it is possible to read out the software.

4.4.1. Commands

All commands can be started by the ARM® processor or via the JTAG interface. Their execution is only restricted by the activated memory protection stored within the SMU. The write sequence for setting up the command can be in any order except that the write to register EXE_CMD (see Table 4.14) must be the last step.

4.4.1.1. Overview

Table 4.9 gives a list of all commands, including when they are allowed to be executed and which flags are influenced internally and in the system management unit. Details are given in subsequent sections.

Table 4.9 List of Commands

Command	Allowed to be Executed	Other Internal Actions
ERASE_MAIN_CMD (00_{HEX})	Always	Flag `allowKey` is set at the end of command execution. Flag `allowBoot` is set at the end of command execution. Flag `allowProg` is set at the end of command execution.
ERASE_BOOT_PROG_CMD (02_{HEX})	Always	Flag `allowKey` is set at the end of command execution. Flag `allowBoot` is set at the end of command execution. Flag `allowProg` is set at the end of command execution.
ERASE_PROG_CMD (03_{HEX})	Always	Flag `allowKey` is set at the end of command execution. Flag `allowProg` is set at the end of command execution.
ERASE_MAINPAGE_CMD (04_{HEX})	BOOT: No lock active PROG: No lock active LOG: Always	---
ERASE_KEY_CMD (06_{HEX})	No lock active or `allowKey` flag set	The protection bits (register SYS_MEMINFO[31:28]; see Table 4.7) are cleared at the end of command execution. The three boundary bit fields in register SYS_MEMINFO in SMU are set to $FFFFFFFF_{HEX}$ at the end of command execution.
UNLOCK_CMD (08_{HEX})	Key-lock only	FAIL: The fail counter (register SYS_MEMINFO[31:30]) in the SMU is incremented at the end of command execution. The permanent lock bit (SYS_MEMINFO[29]) in SMU is set at the end of command execution upon the third failure. SUCCESS: Flag `allowKey` is set at the end of command execution. The key-based lock bit (SYS_MEMINFO[28]) in the SMU is cleared at the end of command execution.
GETENV_CMD (09_{HEX})	Always	The register SYS_MEMINFO will be updated with the extracted values from flash.
WRITE_CMD ($0A_{HEX}$) (also valid for direct write to MAIN area)	BOOT: No lock active or `allowBoot` flag set PROG: No lock active or allowProg flag set LOG: Always	---
SET_KEY_CMD ($0C_{HEX}$)	No lock active	---
SET_BOUNDARY_CMD ($0D_{HEX}$)	No lock active	The three boundary bit fields in register SYS_MEMINFO in SMU are updated at the end of command execution.
LOCK_PERM_CMD ($0E_{HEX}$)	No lock active	The permanent lock bit (SYS_MEMINFO[29]) in SMU is set at the end of command execution.
LOCK_KEY_CMD ($0F_{HEX}$)	No lock active	The key-based lock bit (SYS_MEMINFO[28]) in SMU is set at the end of command execution.

4.4.1.2. ERASE_MAIN_CMD – Erasing the Complete Main Area

This command erases the complete MAIN area of the flash. It can always be executed independently of the lock state (no lock active / key-based lock active / permanent lock active). For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR as well as the bit field `wrSize` of register FC_CMD_SIZE have no meaning. Only the ERASE_MAIN_CMD must be programmed into bit field `cmd` of register FC_CMD_SIZE (see Table 4.13). Then the command execution must be started by writing 1 to bit field `exeCmd` of register FC_EXE_CMD (see Table 4.14).

When the command is started, the bit `coreActive` (FC_STAT_CORE[4]) is set (see Table 4.16). As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the bit `cmdRdy` (FC_STAT_CORE[0]) is set, which is cleared when read. The three flags `allowKey`, `allowBoot` and `allowProg` are also set as the BOOT and PROG sections are now empty. Therefore it is possible to remove the unneeded memory protection by erasing the upper INFO page using ERASE_KEY_CMD and to write new software into the BOOT and PROG sections.

Note: The execution time is approximately 16.3ms.

Note: This command can always be executed via JTAG interface. As the program that the ARM® core is executing might be located in the flash, it is strongly recommended that the following sequence is used:

- The ARM® core first halts the processor via the DHCSR register.
- Then it erases and reprograms the flash.
- Last, it performs a JTAG reset via the SYS_RSTSTAT register in the SMU (see Table 4.8).

Note: This command can always be executed by the ARM® core. To prevent the ARM® core from erasing its own active program, it is strongly recommended that the software is run entirely from RAM memory.

4.4.1.3. ERASE_BOOT_PROG_CMD – Erasing BOOT and PROG section

This command erases the complete BOOT and PROG section of the flash MAIN area, but the content of the LOG section remains in the flash. It can always be executed independent of the lock state (no lock active / key-based lock active / permanent lock active). For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR as well as the bit field `wrSize` of register FC_CMD_SIZE have no meaning. Only the ERASE_BOOT_PROG_CMD must be programmed into bit field `cmd` of register FC_CMD_SIZE. Then the command execution must be started by writing 1 to bit field `exeCmd` of register FC_EXE_CMD.

When the command is started, the bit coreActive (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the bit coreActive (FC_STAT_CORE[4]) is cleared and the bit cmdRdy (FC_STAT_CORE[0]) is set which is cleared when read. The three flags allowKey, allowBoot, and allowProg are set as the BOOT and PROG sections are now empty. Therefore it is possible to remove the unneeded memory protection by erasing the upper INFO page using ERASE_KEY_CMD and to write new software into the BOOT and PROG sections.

Note: The execution time depends on the number of flash pages contained in the BOOT and PROG sections. Each flash page to be erased takes approximately 16.3ms.

Note: This command can always be executed via JTAG interface. As the program that the ARM® core is executing might be located in the flash, it is strongly recommended that the following sequence be used:

- The ARM® core first halts the processor via the DHCSR register.
- Then it erases and reprograms the flash.
- Last, it performs a JTAG reset via SYS_RSTSTAT register in SMU.

Note: This command can always be executed by the ARM® core. To prevent the ARM® core from erasing its own active program, it is strongly recommended that the software is run completely from RAM memory.

Note: Because this command erases all flash pages from the first page until the page in front of the flash page logStart, this command must only be used when the boundaries have a meaningful setting (see Table 4.7).

4.4.1.4. ERASE_PROG_CMD – Erasing PROG section

This command erases the complete PROG section of the flash MAIN area, but the content of the BOOT and the LOG sections remains in the flash memory. It can always be executed independently of the lock state (no lock active / key-based lock active / permanent lock active). For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR as well as the bit field wrSize of register FC_CMD_SIZE have no meaning. Only the ERASE_PROG_CMD must be programmed into the bit field cmd of register FC_CMD_SIZE. Then the command execution must be started by writing 1 to bit field exeCmd of register FC_EXE_CMD.

When the command is started, the coreActive bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the bit coreActive (FC_STAT_CORE[4]) is cleared and the cmdRdy bit (FC_STAT_CORE[0]) is set, which is cleared when read. The two flags allowKey and allowProg are also set as the PROG section is now empty. Therefore the unneeded memory protection can be removed by erasing the upper INFO page using ERASE_KEY_CMD and writing new software into the PROG sections.

Note: The execution time depends on the number of flash pages contained in PROG section. Each flash page to be erased takes approximately 16.3ms.

Note: This command can always be executed via JTAG interface. As the program that the ARM® core is executing might be located in the flash, it is strongly recommended that the following sequence is used:

- The ARM® core first halts the processor via the DHCSR register.
- Then it erases and reprograms the flash.
- Last, it performs a JTAG reset via the SYS_RSTSTAT register in SMU.

Note: This command can always be executed by the ARM® core. To prevent the ARM® core from erasing its own active program, it is strongly recommended that the software is run entirely from RAM memory.

Important: As this command erases all flash pages from the flash page `progStart` through the page in front of the flash page `logStart`, this command must only be used when the boundaries have a meaningful setting (see Table 4.7).

4.4.1.5. ERASE_MAINPAGE_CMD – Erasing a Single Page in the MAIN Area

This command erases a single page within the flash MAIN area. Its execution depends on the memory protection and to which section of the MAIN area (BOOT / PROG / LOG) the page to be erased belongs. When no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}), the selected flash page will be erased independent of its location. If any lock is active, the page will only be erased if it is located within the LOG section. Otherwise, the command is rejected.

For command execution, the settings of register FC_RAM_ADDR, as well as the `wrSize` bit field of register FC_CMD_SIZE, have no meaning. The register FC_FLASH_ADDR (see Table 4.12) is used to select the flash page to be erased. The value to be programmed must be calculated by shifting the flash address pointing to any location in the desired page two bits to the right. The ERASE_MAINPAGE_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. Then the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because a page within BOOT or PROG section was selected while the memory protection is active, the `invalidArea` bit (FC_STAT_CORE[2]) is also set.

Note: The execution time is approximately 16.3ms if the command is not rejected.

Warning: This command is intended for erasing a page in the LOG section. Special care must be taken for erasing a page within the BOOT or PROG section as software might become inconsistent.

Note: Because this command distinguishes between the LOG section on one side and the BOOT and PROG section on the other if memory protection is active using the setting of `logStart`, this command must only be used if the boundaries have a meaningful setting in this case.

4.4.1.6. ERASE_KEY_CMD – Erasing the Upper INFO Page

This command erases the upper INFO page, which contains the lock information, the memory boundaries, and the key to be used for the key-based lock. It is only executed when no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}) or when the `allowKey` flag (FC_STAT_CORE[5]) is set. Otherwise, the command is rejected. For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR, as well as the `wrSize` bit field of register FC_CMD_SIZE, have no meaning. Only the ERASE_KEY_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. After that, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the `allowKey` flag was not set but the memory protection is active, the `invalidCmd` bit (FC_STAT_CORE[1]) is also set. If the command was not rejected, all three boundary fields within register SYS_MEMINFO (bits [26:0]) in the SMU are set to all 1s and the `protInfo` bit field is set to all 0s at end of the command execution.

Note: The execution time is approximately 16.3ms if the command is not rejected.

Note: The register SYS_MEMINFO is updated at the end of the successful execution of this command.

Important: In addition to the lock and the key information, the boundaries are erased. Therefore the user must reprogram the boundaries afterward using the SET_BOUNDARY_CMD (see section 4.4.1.8). Boundaries can be read and stored before command execution from register SYS_MEMINFO.

4.4.1.7. WRITE_CMD – Writing 1 to 32 Words from RAM to Flash

This command copies up to 32 words from RAM into the flash MAIN area. Its execution depends on the memory protection state, the destination section of the flash, and the "allow" flags set in register FC_STAT_CORE (see Table 4.16).

- If the write will be performed to the BOOT section, the command is only executed when no lock is active or when the `allowBoot` flag is set. Otherwise the command is rejected.
- If the write will be performed to the PROG section, the command is only executed when no lock is active or when the "`allowProg`" flag is set. Otherwise the command is rejected.
- If the write will be performed to the LOG section, the command is always executed.

First, the data to be stored into the flash must be written into the RAM. The data must be placed word-aligned into the RAM with consecutive words to consecutive addresses. The register FC_RAM_ADDR must be written with the RAM address where the word containing the first word was stored (see Table 4.11). The value to be programmed must be calculated by shifting the RAM address pointing to that location by two bits to the right. Register FC_FLASH_ADDR must be written with the flash address where the first word will be stored (see Table 4.12). The value to be programmed must be calculated by shifting the flash address pointing to that location by two bits to the right. Special care must be taken when more than one byte will be written. While there is no restriction on how the block of words is placed into the RAM (no block alignment), the WRITE_CMD only operates on a single flash row (one flash page consists of four flash rows; one row consists of 32 words). When a row boundary is reached during execution of a write command while there is still data to be written present, the flash address wraps to the beginning of the row.

The register FC_CMD_SIZE must be programmed with the WRITE_CMD to the `cmd` bit field and the number of bytes to be written to the `wrSize` bit field. For `wrSize` equal to 00001_{BIN}, one word is written; for "`wrSize`" equal to 00010_{BIN}, two words are written, and so on. A value of 00000_{BIN}, is interpreted as 32 words, which means that a complete row must be programmed. After all registers are set appropriately, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the memory protection is active and the corresponding "allow" flag is not set, the `invalidArea` bit (FC_STAT_CORE[2]) is also set.

Note: The execution time is approximately (22 + 50 * N)µs where N is the number of words to be programmed.

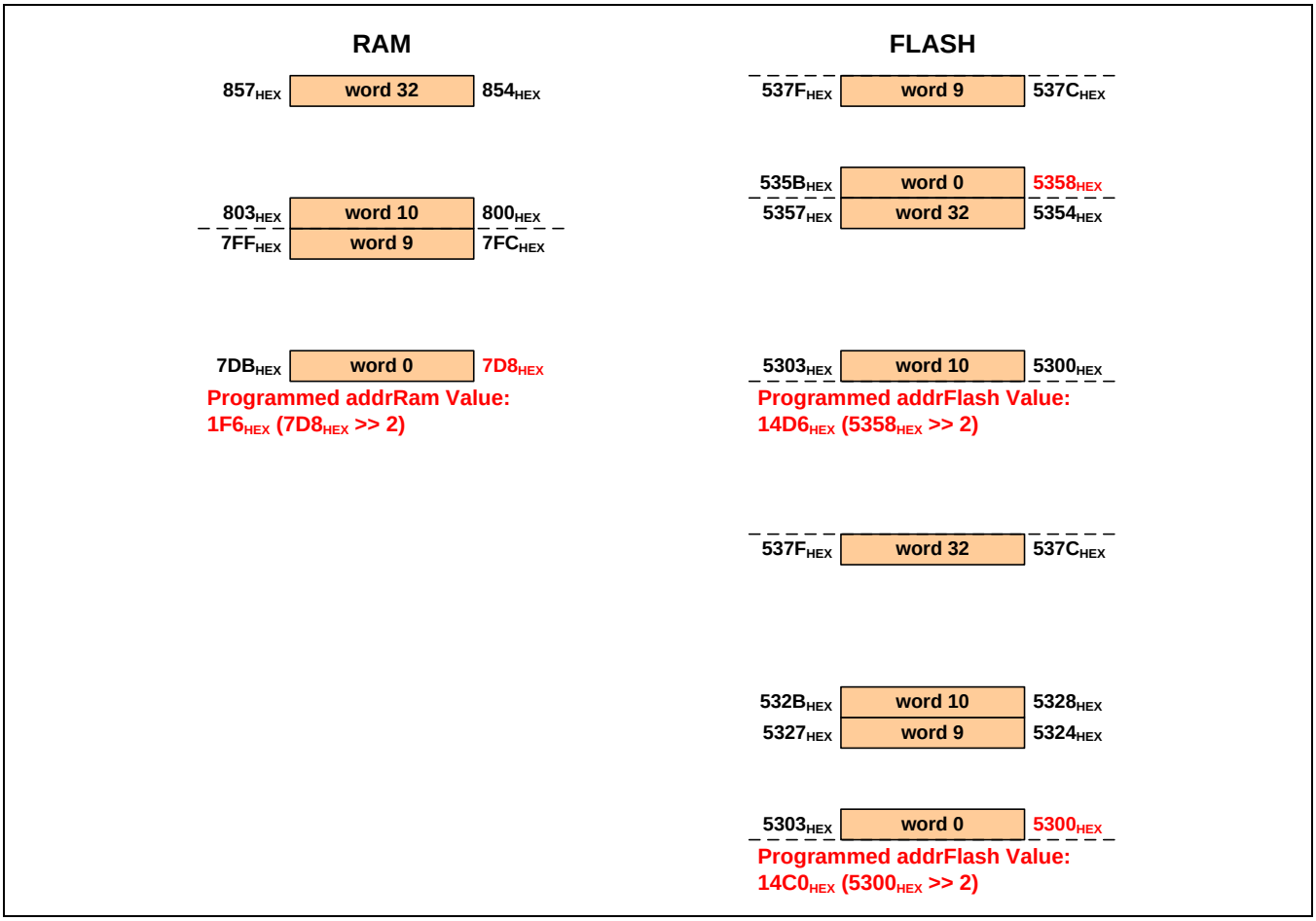
Note: The value to be programmed into register FC_RAM_ADDR does not contain the last two digits (always 0). For this, the RAM address must be shifted right by two bits (value = (RAM address >> 2).

Note: The value to be programmed into register FC_FLASH_ADDR does not contain the last two digits (always 0). For this, the flash address must be shifted right by two bits (value = (flash address >> 2).

Note: This command does not erase the locations to be written. Therefore software must ensure that the locations are empty.

Note: This command writes only within a single flash row. The address wraps at the end of a flash row back to the beginning of the row and does not continue in the next row.

Figure 4.5 Block Writes Examples: from RAM to Flash with/without Wrapping at the Flash Row Boundary



4.4.1.8. SET_BOUNDARY_CMD – Setting the Three Boundaries

This command stores the required memory boundaries (`progStart`, `logStart`, `ramSplit`) from RAM into upper INFO page at address 210_{HEX}. It is only executed when no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}). Otherwise, the command is rejected. For command execution, the settings of register FC_FLASH_ADDR, as well as the bit field `wrSize` of register FC_CMD_SIZE have no meaning. First, one word containing the three boundaries must be written into the RAM (word aligned) in the same format as they are stored into the flash or as they are stored in register SYS_MEMINFO:

- word bits [7:0] = flashAddr[16:9]; (`progStart`; number of first flash page of PROG section)
- word bits [15:8] = flashAddr[16:9]; (`logStart`; number of first flash page for LOG section)
- word bits [26:16] = ramAddr[12:2]; (`ramSplit`; first accessible word when any lock is active)
- word bits [31:27] are “don’t care”

The register FC_RAM_ADDR must be written with the RAM address where the word containing the three boundaries was stored (see Table 4.11). The value to be programmed must be calculated by shifting the RAM address pointing to that location by two bits to the right. The SET_BOUNDARY_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE (see Table 4.13). After that, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the memory protection is active, the `invalidCmd` bit (FC_STAT_CORE[1]) is also set. If the command was not rejected, all three boundary fields within register SYS_MEMINFO in SMU are set to the programmed values at end of the command execution.

Note: The execution time is approximately 72µs when the command is not rejected.

Note: The value to be programmed into register FC_RAM_ADDR does not contain the last two digits (always 0). For this, the RAM address must be shifted right by two bits (value = (RAM address >> 2).

Note: This command does not erase the upper INFO page. Therefore software must ensure that the location where the boundaries are stored is empty. If the location already contains boundary information, the ERASE_KEY_CMD must be executed in advance.

4.4.1.9. SET_KEY_CMD – Storing the Key for the Key-Based Lock

This command stores the key required for the key-based lock from RAM into the upper INFO page at address 300_{HEX} and following. It is only executed when no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}). Otherwise, the command is rejected. For command execution, the settings of register FC_FLASH_ADDR, as well as the `wrSize` bit field of register FC_CMD_SIZE, have no meaning. First, the key must be stored into the RAM using the format shown in Table 4.10. The key has a selectable length of 1 to 32 bytes. The key length is contained in the first key word. For a key length of 1, the key consists of key word 0 only; for a key length of 2, the key consists of key word 0 and key word 1, and so on. A key length of 0 is interpreted as 32, which means that N is 31.

Table 4.10 Key Format

Address Offset	Bits 31:5	Bits 4:0
0	Key word 0	Key length
1	Key word 1	
…	…	
N (N<= 31)	Key word N	

The register FC_RAM_ADDR must be written with the RAM address where the key word 0, which contains the key length, was stored. The value to be programmed must be calculated by shifting the RAM address pointing to that location by two bits to the right. The SET_KEY_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. Next, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the coreActive bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the coreActive bit (FC_STAT_CORE[4]) is cleared and the bit cmdRdy (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the memory protection is active, the invalidCmd bit (FC_STAT_CORE[1]) is also set.

Note: Storing the key does not activate the key-based lock. For activation, the LOCK_KEY_CMD must be executed (see section 4.4.1.11).

Note: After the key is stored but before the lock is activated, it is possible to read the key from the upper INFO page. This might be needed to check that the programming was correct.

Note: The execution time depends on the number of key words. Approximately 72µs is needed for each key word to be stored.

Note: The value to be programmed into register FC_RAM_ADDR does not contain the last two digits (always 0). For this, the RAM address must be shifted right by two bits (value = (RAM address >> 2).

Note: This command does not erase the upper INFO page. Therefore software must ensure that the locations where the key words are stored are empty. If the locations already contain a key, the ERASE_KEY_CMD must be executed in advance.

4.4.1.10. LOCK_PERM_CMD – Activation of Permanent Lock

This command stores the value 0000 0000_{HEX} at address 208_{HEX} in the upper INFO page, which activates the permanent lock. It is only executed when no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}). Otherwise, the command is rejected. For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR as well as the wrSize bit field of register FC_CMD_SIZE have no meaning. Only the LOCK_PERM_CMD must be programmed into the cmd bit field of register FC_CMD_SIZE. After that, the command execution must be started by writing 1 to the exeCmd bit field of register FC_EXE_CMD.

When the command is started, the coreActive bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the coreActive bit (FC_STAT_CORE[4]) is cleared and the cmdRdy bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the memory protection is active, the invalidCmd bit (FC_STAT_CORE[1]) is also set. If the command was not rejected, the permanent lock bit (SYS_MEMINFO[29]) in the SMU is set to 1 at end of the command execution.

Note: The execution time is approximately 72µs if the command is not rejected.

Note: Make sure that the boundaries have been stored before execution of this command as they cannot be programmed afterward.

Note: This lock can only be removed by first executing an appropriate erase command (ERASE_MAIN_CMD, ERASE_BOOT_PROG_CMD, ERASE_PROG_CMD), which guarantees that the program section is empty, and then executing the ERASE_KEY_CMD (see section 4.4.1.6).

© 2016 Integrated Device Technology, Inc.

Note: Reprogramming is possible by first executing an appropriate erase command (ERASE_MAIN_CMD, ERASE_BOOT_PROG_CMD, ERASE_PROG_CMD) to get a temporary access due to set “allow” flags and then performing the correct amount of WRITE_CMD commands. After a reset is applied or the “allow” flags are cleared, the permanent lock is reactivated.

4.4.1.11. LOCK_KEY_CMD – Activation of Key-based Lock

This command stores the value 0000 0000_{HEX} at address $20C_{HEX}$ within the upper INFO page, which activates the key-based lock. It is only executed when no lock is active (SYS_MEMINFO[29:28] == 00_{BIN}). Otherwise, the command is rejected. For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR as well as the `wrSize` bit field of register FC_CMD_SIZE have no meaning. Only the LOCK_KEY_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. After that, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because the memory protection is active, the `invalidCmd` bit (FC_STAT_CORE[1]) is also set. If the command was not rejected, the key-based lock bit (SYS_MEMINFO[28]) in the SMU is set to 1 at the end of the command execution.

Note: The execution time is approximately 72µs if the command is not rejected.

Important: Ensure that the boundaries and the key have been stored before execution of this command as they cannot be programmed afterward.

Note: This lock can be removed by first executing an appropriate erase command (ERASE_MAIN_CMD, ERASE_BOOT_PROG_CMD, ERASE_PROG_CMD), which guarantees that the program section is empty, and then executing the ERASE_KEY_CMD (see section 4.4.1.6). It can also be removed by successful execution of the UNLOCK command (section 4.4.1.12). This is used to get access to the software as no erase is performed.

Note: Reprogramming is possible by first executing an appropriate erase command (ERASE_MAIN_CMD, ERASE_BOOT_PROG_CMD, ERASE_PROG_CMD) to get a temporary access to set “allow” flags and then performing the required WRITE_CMD commands. After a reset is applied or the “allow” flags are cleared, the key-based lock is reactivated.

4.4.1.12. UNLOCK_CMD – Deactivation of the Key-Based Lock

This command is used to temporarily deactivate the key-based lock to get access to the BOOT and PROG section. It is only executed when the key-based lock is active (SYS_MEMINFO[29:28] == 01_{BIN}). Otherwise, the command is rejected. For command execution, the settings of register FC_FLASH_ADDR, as well as the `wrSize` bit field of register FC_CMD_SIZE, have no meaning. First, the key must be stored in the RAM in the same manner as for programming the key using the format shown in Table 4.10. The register FC_RAM_ADDR must be written with the RAM address where the key word 0 was stored. The value to be programmed must be calculated by shifting the RAM address pointing to that location by two bits to the right. The UNLOCK_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. Next, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write) as well as writes to all registers of the flash controller are postponed. When the software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read. If the command execution was rejected because no lock or the permanent lock is active, the bit `invalidCmd` (FC_STAT_CORE[1]) is also set. If the unlock procedure failed due to a wrong key, the `unlockFail` bit (FC_STAT_CORE[3]) is set, the failure counter (SYS_MEMINFO[31:30]) is incremented, and, if it was the third failure, the permanent lock is activated (SYS_MEMINFO[29] set to 1). If the unlock procedure was successful, the key-based lock is deactivated (SYS_MEMINFO[28] set to 0).

Important: Although the protection is removed in register SYS_MEMINFO, the protection remains in the upper flash INFO page causing a temporary inconsistency between the register settings and the flash content. To permanently remove the lock after the successful execution, an ERASE_KEY_CMD must be executed afterward (see section 4.4.1.6). Otherwise the key-based lock will be reactivated after a reset is applied or a GETENV_CMD is executed.

4.4.1.13. GETENV_CMD – Restoring Memory Protection

This command restores the memory protection and boundary information from the upper flash INFO page into the register SYS_MEMINFO in the SMU. Although it can always be executed independent of the lock state, it is only required to be performed after the successful execution of the UNLOCK_CMD when the lock will be reactivated. For command execution, the settings of registers FC_RAM_ADDR and FC_FLASH_ADDR, as well as the `wrSize` bit field of register FC_CMD_SIZE, have no meaning. Only the GETENV_CMD must be programmed into the `cmd` bit field of register FC_CMD_SIZE. Next, the command execution must be started by writing 1 to the `exeCmd` bit field of register FC_EXE_CMD.

When the command is started, the `coreActive` bit (FC_STAT_CORE[4]) is set. As long as the command is active, all direct accesses to the flash (read and write), as well as writes to all registers of the flash controller, are postponed. When software is running from RAM, the system is still able to run. Read accesses to the registers of the flash controller are allowed while the command is active to check the status of the command execution. When the command has finished, the `coreActive` bit (FC_STAT_CORE[4]) is cleared and the `cmdRdy` bit (FC_STAT_CORE[0]) is set, which is cleared when read.

4.4.2. Register Overview for Flash Controller

4.4.2.1. Register "FC_RAM_ADDR" – RAM Address for Command Execution

Table 4.11 Register FC_RAM_ADDR – system address 4000 0800$_{HEX}$

Name	Bits	Default	Access	Description
addrRam	[10:0]	0000 0000$_{HEX}$	RW	RAM word address where data to be written into flash is located (first address); hardware increments this address when more than one word is to be written. Commands requiring this register: SET_KEY_CMD SET_BOUNDARY_CMD WRITE_CMD UNLOCK_CMD **Note:** The last two address digits of the RAM are not included as they are always 0. For programming, the RAM address must be shifted right by two bits.
Unused	[31:11]		RO	Unused; always write as 0.

4.4.2.2. Register "FC_FLASH_ADDR" – FLASH Address for Command Execution

Table 4.12 Register FC_FLASH_ADDR – system address 4000 0804$_{HEX}$

Name	Bits	Default	Access	Description
flashAddr	[14:0]	0000 0000$_{HEX}$	RW	Flash word address (first address) where data will be written; also used as the pointer to the page to be erased. Commands requiring this register: WRITE_CMD ERASE_MAINPAGE_CMD **Note:** The last two address digits of the flash are not included as they are always 0. For programming, the flash address must be shifted right by two bits.
Unused	[31:15]		RO	Unused; always write as 0.

4.4.2.3. Register "FC_CMD_SIZE" – Command Setup and Write Size Configuration

Table 4.13 Register FC_CMD_SIZE – system address 4000 0808$_{HEX}$

Name	Bits	Default	Access	Description
cmd	[3:0]	0000 0000$_{HEX}$	RW	Command to be executed by the flash controller. Valid commands: 0_{HEX}: ERASE_MAIN_CMD 2_{HEX}: ERASE_BOOT_PROG_CMD 3_{HEX}: ERASE_PROG_CMD 4_{HEX}: ERASE_MAINPAGE_CMD 6_{HEX}: ERASE_KEY_CMD 8_{HEX}: UNLOCK_CMD 9_{HEX}: GETENV_CMD A_{HEX}: WRITE_CMD C_{HEX}: SET_KEY_CMD D_{HEX}: SET_BOUNDARY_CMD E_{HEX}: LOCK_PERM_CMD F_{HEX}: LOCK_KEY_CMD
Unused	[7:4]		RO	Unused; always write as 0.
wrSize	[12:8]		RW	Number of words to be written to the flash; 0 is interpreted as 32. **Note:** Writing is always performed within a row. 1 row contains 32 words. While the RAM address is always incremented, the flash address wraps at the row boundary to the beginning of the row. It is in the responsibility of the user to take care of this.
Unused	[31:13]		RO	Unused; always write as 0.

4.4.2.4. Register "FC_EXE_CMD" – Start Command Execution

Table 4.14 Register FC_EXE_CMD – system address 4000 080C$_{HEX}$

Name	Bits	Default	Access	Description
exeCmd	[0]	0000 0000$_{HEX}$	RW	Writing 1 to this bit starts the execution of the configured command; always read as 0.
Unused	[31:1]		RO	Unused; always write as 0.

4.4.2.5. Register "FC_IRQ_EN" – Interrupt Enables

Table 4.15 Register FC_IRQ_EN – system address 4000 0810$_{HEX}$

Name	Bits	Default	Access	Description
enIrq0	[0]	0_{BIN}	RW	When set to 1, the status signal `cmdRdy` is allowed to drive the interrupt line.
enIrq1	[1]	0_{BIN}	RW	When set to 1, the status signal `invalidCmd` is allowed to drive the interrupt line.
enIrq2	[2]	0_{BIN}	RW	When set to 1, the status signal `invalidArea` is allowed to drive the interrupt line.
enIrq3	[3]	0_{BIN}	RW	When set to 1, the status signal `unlockFail` is allowed to drive the interrupt line.
enIrq4	[4]	0_{BIN}	RW	When set to 1, the status signal `dataAll1` is allowed to drive the interrupt line.
enIrq5	[5]	0_{BIN}	RW	When set to 1, the status signal `data1Err` is allowed to drive the interrupt line.
enIrq6	[6]	0_{BIN}	RW	When set to 1, the status signal `data2Err` is allowed to drive the interrupt line.
enIrq7	[7]	0_{BIN}	RW	When set to 1, the status signal `prog1Err` is allowed to drive the interrupt line.
Unused	[31:8]	00 0000$_{HEX}$	RO	Unused; always write as 0.

4.4.2.6. Register "FC_STAT_CORE" – FLASH Controller Core Status

Table 4.16 Register FC_STAT_CORE – system address 4000 0814$_{HEX}$

Name	Bits	Default	Access	Description
cmdRdy	[0]	0000 0000$_{HEX}$	RC	This bit is set when a command execution has finished; it is cleared when this register is read.
invalidCmd	[1]		RC	This bit is set when an invalid command was executed; it is cleared when this register is read.
invalidArea	[2]		RC	This bit is set when a command is executed targeting a protected area; e.g., performing ERASE_MAINPAGE_CMD to program space when the flash is locked. It is cleared when this register is read.
unlockFail	[3]		RC	This bit is set when the UNLOCK_CMD fails; it is cleared when this register is read.
coreActive	[4]		RO	This bit reflects the status of the core state machine; when set, the core is active.
allowKey	[5]		RO	When set but flash is locked, the ERASE_KEY_CMD is allowed to be performed.
allowBoot	[6]		RO	When set but flash is locked, the WRITE_CMD is allowed to be performed on the boot space.
allowProg	[7]		RO	When set but flash is locked, the WRITE_CMD is allowed to be performed on the program space.
clrAllow	[8]		W1C	Writing 1 to this bit clears all 3 allow flags.
Unused	[31:9]		RO	Unused; always write as 0.

4.4.2.7. Register "FC_STAT_PROG" – FLASH Controller Instruction Fetch Status

Table 4.17 Register FC_STAT_PROG – system address 4000 0818$_{HEX}$

Name	Bits	Default	Access	Description
addrProg	[17:0]	0000 0000$_{HEX}$	RO	This register contains the address of the instruction fetch error that caused the first of the three flags below to be set; the highest bit is used to distinguish between MAIN (0) and INFO (1) area.
Unused	[28:18]		RO	Unused; always read as 0.
progAll1	[29]		RC	This bit is set when an instruction fetch occurs to an erased memory address; it is cleared when this register is read.
prog1Err	[30]		RC	This bit is set when a correctable error occurs during an instruction fetch; it is cleared when this register is read.
prog2Err	[31]		RC	This bit is set when an uncorrectable error occurs during an instruction fetch; it is cleared when this register is read.

4.4.2.8. Register "FC_STAT_DATA" – FLASH Controller Data Load Status

Table 4.18 Register FC_STAT_DATA – system address 4000 081C$_{HEX}$

Name	Bits	Default	Access	Description
addrData	[17:0]	0000 0000$_{HEX}$	RO	This register contains the address of the read error that caused the first of the three flags below to be set; the highest bit is used to distinguish between MAIN (0) and INFO (1) area.
Unused	[28:18]		RO	Unused; always read as 0.
dataAll1	[29]		RC	This bit is set when a read is performed to an erased memory address; it is cleared when this register is read.
data1Err	[30]		RC	This bit is set when a correctable error occurs during a read; it is cleared when this register is read.
data2Err	[31]		RC	This bit is set when an uncorrectable error occurs during a read; it is cleared when this register is read.

4.5. GPIO

There are 16 GPIO pads (GPIO00 to GPIO15) implemented in the MCU, but only the lower five (GPIO00 – GPIO04) are bonded out of the package. The unbonded GPIO pads contain a pull-up resistor in their pad and must never be used (must be kept in their default state as input). The bonded GPIO pads contain pull-down resistors.

Note: Do not use unbonded GPIO pads. Keep them in their reset state where they are configured as inputs.

Each GPIO pad can be individually configured to operate as an input or output. When configured as an output, the value driven out of the GPIO pad can be directly written or controlled via a set-clear register. Additionally, each GPIO can be enabled to be used as a trigger source for the 32-bit timer or it can be used as an interrupt source with a selectable edge.

Note: Each register in the GPIO module can be accessed on byte, half-word and word size.

4.5.1. Normal Functionality

When a GPIO pad should be used as a simple input or output, registers GPIO_DIR, GPIO_IN, GPIO_OUT and GPIO_SETCLR are needed. Register GPIO_DIR is used to select the direction of the GPIO pad (see Table 4.19). By default, the GPIO pad is configured as an input pad. The synchronized value from that pad can be read by reading register GPIO_IN (see Table 4.20). The values from those GPIO pads that are configured as outputs or which have other functionality will be ignored.

The value driven out of a GPIO pad that is configured as an output can be set by writing to register GPIO_OUT (see Table 4.21). The initial value can already be written before switching the direction from input to output. Instead of writing all output values in each access, it is also possible to set and clear dedicated output values by using the register GPIO_SETCLR (see Table 4.22). This functionality avoids needing to read and modify the GPIO_OUT register when only some bits need to be changed.

Note: In addition to the internal configuration, the settings of register SYS_MEMPORTCFG (see Table 4.6) in the SMU module must be taken into account. This register is used to configure the output type (push-pull or open-drain (default)) and to map sub-modules of the ZSYSTEM2 module onto GPIO pads. When the latter is true, normal GPIO functionality is not available.

4.5.2. Trigger Functionality

Each GPIO pad can be used as an external trigger source for the 32-bit timer (see section 4.6). To enable the trigger functionality, the GPIO pad must be configured as an input and the trigger functionality must be enabled via register GPIO_TRIGEN (see Table 4.26). It is possible to enable several GPIO pads as external trigger sources; however it is not recommended because power consumption slightly increases when trigger functionality is enabled.

Note: It is not sufficient to enable a GPIO pad as a trigger source. The 32-bit timer must also be configured appropriately, and the desired GPIO trigger source must be selected via register T32_TRIGSEL (see Table 4.29) within the 32-bit timer.

4.5.3. Interrupt Functionality

Each GPIO pad can be used as an external, edge-sensitive interrupt source. To enable the interrupt functionality, the GPIO pad must be configured as an input and the interrupt functionality must be enabled via register GPIO_IRQEN. Additionally, it is selectable via register GPIO_IRQEDGE whether a rising or a falling edge on the GPIO pad activates the interrupt (see Table 4.25).

All GPIO pads enabled as external interrupt sources drive one single interrupt line connected to ARM® interrupt 5. The user can determine which GPIO pad caused the interrupt by reading register GPIO_IRQSTAT. All interrupt status bits are cleared when reading register GPIO_IRQSTAT (see Table 4.23).

Note: As the synchronization flip-flops are only continuously clocked when the trigger or interrupt functionality is enabled for the corresponding GPIO pad, it might be possible that an unwanted interrupt occurs when enabling the interrupt functionality. To avoid this, the following sequence must be guaranteed by software:

- Enable the GPIO trigger functionality and select the desired interrupt edge to be used.
- Enable the GPIO interrupt functionality at least three cycles after enabling as a trigger.
- Disable the GPIO trigger functionality (when not needed in parallel).

© 2016 Integrated Device Technology, Inc.

4.5.4. Register Overview of GPIO Module

4.5.4.1. Register "GPIO_DIR" – GPIO Direction

Table 4.19 Register GPIO_DIR – system address 4000 1400$_{HEX}$

Name	Bits	Default	Access	Description
gpioDir	[15:0]	0000_{HEX}	RW	Direction of each GPIO. 1: GPIO pad is switched as an output 0: GPIO pad is switched as an input
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.2. Register "GPIO_IN" – GPIO Input Value

Table 4.20 Register GPIO_IN – system address 4000 1404$_{HEX}$

Name	Bits	Default	Access	Description
gpioIn	[15:0]	0000_{HEX}	RO	Synchronized input value.
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.3. Register "GPIO_OUT" – GPIO Output Value

Table 4.21 Register GPIO_OUT – system address 4000 1408$_{HEX}$

Name	Bits	Default	Access	Description
gpioOut	[15:0]	0000_{HEX}	RW	Value to be driven out of each GPIO; can be selected individually.
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.4. Register "GPIO_SETCLR" – Set and Clear for GPIO Output Value

Table 4.22 Register GPIO_SETCLR – system address 4000 140C$_{HEX}$

Name	Bits	Default	Access	Description
15 : 0	[15:0]	0000_{HEX}	WO	There is one set bit per GPIO; the value driven out of the GPIO is set to 1 when 1 is written to the corresponding bit (lower priority than clear); always read as 0.
31 : 16	[31:16]	0000_{HEX}	WO	There is one set bit per GPIO; the value driven out of the GPIO is set to 0 when 1 is written to the corresponding bit (higher priority than set); always read as 0.

© 2016 Integrated Device Technology, Inc.

4.5.4.5. Register "GPIO_IRQSTAT" – Interrupt Status

Table 4.23 Register GPIO_IRQSTAT – system address 4000 1410$_{HEX}$

Name	Bits	Default	Access	Description
irqStat	[15:0]	0000_{HEX}	RC	This register reflects the interrupt status of each GPIO that is enabled as an interrupt.
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.6. Register "GPIO_IRQEN" – Interrupt Enable

Table 4.24 Register GPIO_IRQEN – system address 4000 1414$_{HEX}$

Name	Bits	Default	Access	Description
irqEn	[15:0]	0000_{HEX}	RW	When set to 1, the corresponding interrupt is allowed to drive the interrupt line when the appropriate edge occurs.
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.7. Register "GPIO_IRQEDGE" – Edge Selection for Interrupt

Table 4.25 Register GPIO_IRQEDGE – system address 4000 1418$_{HEX}$

Name	Bits	Default	Access	Description
irqEdge	[15:0]	0000_{HEX}	RW	0: a rising edge on the corresponding GPIO triggers the IRQN line 1: a falling edge on the corresponding GPIO triggers the IRQN line
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

4.5.4.8. Register "GPIO_TRIGEN" – Trigger Enable

Table 4.26 Register GPIO_TRIGEN – system address 4000 141C$_{HEX}$

Name	Bits	Default	Access	Description
trigEn	[15:0]	0000_{HEX}	RW	When set to 1, the corresponding GPIO drives its trigger line.
Unused	[31:16]	0000_{HEX}	RO	Unused; always write as 0.

© 2016 Integrated Device Technology, Inc.

4.6. 32-Bit Timer

The timer provides event counting on the rising clock edge with a 32-bit resolution. It is capable of counting clock events in Timer Mode and counting events from a selectable external trigger signal in Counter Mode. The external trigger can be configured to operate on the rising or falling edges as well as on the low or high level. Additionally, it can be selected whether the timer/counter stops when it overflows or continues its operation.

The timer has an interrupt line that is active-high and set high for a single clock cycle whenever the counter overflows. The interrupt line is connected to ARM® interrupt 4.

Note: The counter is incremented when enabled.

4.6.1. Timer Mode

In Timer Mode (bit `modeTC` in register `T32_CTRL` == 0), the counter register is incremented in each clock cycle (see Table 4.28). When the counter reaches FFFF $FFFF_{HEX}$, the reload value is copied into the counter register and both the overflow bit and the interrupt line are set high for one clock cycle. When Reload Mode is enabled (`modeSR` == 0), the counter continues counting. Otherwise the counter stops. The two other control bits (`modeLE` and `modePN`) have no meaning in this mode.

4.6.2. Counter Mode

In Counter Mode (bit `modeTC` in register `T32_CTRL` == 0), the counter register is incremented in each clock cycle when the trigger is active. When the counter has a value of FFFF $FFFF_{HEX}$ and the trigger is active, the reload value is copied into the counter register and both the overflow bit and the interrupt line are set high for one clock cycle. When Reload Mode is enabled (`modeSR` == 0), the counter continues counting. Otherwise the counter stops. The two control bits `modeLE` and `modePN` are used to configure the trigger as shown in Table 4.27.

Table 4.27 Configuration of Trigger Behavior

modeLE	modePN	Behavior
0	0	The trigger is active when the trigger input is low but was high in the previous clock cycle (sensitive on falling edge).
0	1	The trigger is active when the trigger input is high but was low in the previous clock cycle (sensitive on rising edge).
1	0	The trigger is active when the trigger input is low (sensitive on low level).
1	1	The trigger is active when the trigger input is high (sensitive on high level).

4.6.3. Timer Module Register Overview

4.6.3.1. Register "T32_CTRL" – Timer Control

Table 4.28 Register T32_CTRL – system address 4000 1000$_{HEX}$

Name	Bits	Default	Access	Description
en	[0]	0000 0000$_{HEX}$	RW	Enable bit for timer; this bit is cleared by hardware when an overflow occurs and the module is operating in Single-Shot Mode.
modeTC	[1]		RW	Select between the timer and counter modes: 0: Timer Mode 1: Counter Mode
modeSR	[2]		RW	Select between the reload and single-shot modes. 0: Reload Mode; at overflow, the reload value is copied into the counter register and the counter continues 1: Single-Shot Mode; at overflow, the reload value is copied into the counter register and the counter stops
modeLE	[3]		RW	Select between level or edge sensitive trigger; Counter Mode only. 0: Trigger is edge-sensitive 1: Trigger is level-sensitive
modePN	[4]		RW	Selects between the rising or falling edge active trigger (modeLE == 1) or high or low level (modeLE == 0); Counter Mode only. 0: Trigger on the falling edge / low level 1: Trigger on the rising edge / high level
overflow	[5]		RO	Overflow flag (strobe); set for a single cycle when the counter overflows; this bit also drives the interrupt line.
Unused	[31:6]		RO	Unused; always write as 0.

4.6.3.2. Register "T32_TRIGSEL" – Trigger Selection

Table 4.29 Register T32_TRIGSEL – system address 4000 1004$_{HEX}$

Name	Bits	Default	Access	Description
trigSel	[4:0]	0000 0000$_{HEX}$	RW	Select signal for the trigger source. 00000_{BIN}: No trigger source 00001_{BIN}: GPIO00 is used as trigger source 00010_{BIN}: GPIO01 is used as trigger source … 10000_{BIN}: GPIO15 is used as trigger source 10001_{BIN} to 11111_{BIN}: no trigger source
Unused	[31:5]		RO	Unused; always write as all 0s.

4.6.3.3. Register “T32_CNT” – Timer Value

Table 4.30 Register T32_CNT – system address 4000 1008$_{HEX}$

Name	Bits	Default	Access	Description
counter	[31:0]	0000 0000$_{HEX}$	RW	Timer value; this register can be written directly whether or not the timer is enabled. It is set to the reload value when reload value is written.

4.6.3.4. Register “T32_REL” – Timer Reload Value

Table 4.31 Register T32_REL – system address 4000 100C$_{HEX}$

Name	Bits	Default	Access	Description
reloadVal	[31:0]	0000 0000$_{HEX}$	RW	Timer reload value; when the timer (counter) overflows, the reload value is copied into the counter register. In Reload Mode, the timer continues; when Reload Mode is not enabled, the timer stops.

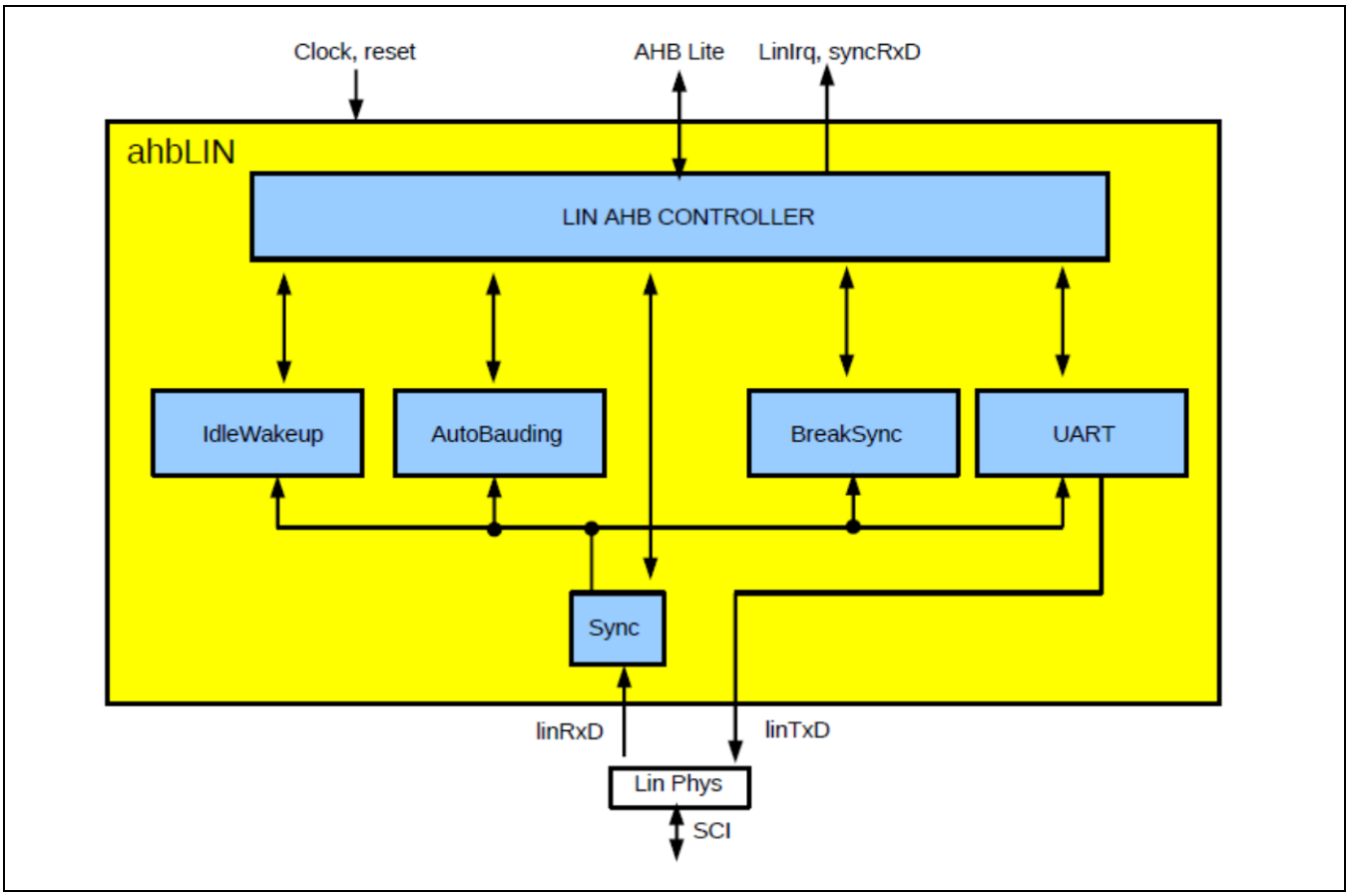
4.7. LIN Communication Control Logic (ahbLIN)

Although the LIN PHY is integrated in the SBC (see section 3.9), LIN communication is controlled by the ahbLIN block (the “LIN UART” block in Figure 2.2) on the MCU side (ahb stands for the Advanced High-performance Bus widely used on ARM® microcontrollers). The ahbLIN has access to the TXD and RXD lines going to the LIN PHY on the SBC.

The ahbLIN module is the communication control logic that performs the serial communication interface (SCI) according the *LIN Protocol Specification (Revision 2.2, SAE J2602)*. The type of the interface is LIN slave.

The LIN communication logic has a modular structure of different independent transceiver processes. The processes are driven by the synchronized serial data stream linRxD and linTxD and are controlled by the LIN ahb controller module.

Figure 4.6 ahbLIN Block Diagram



Key Features:

- Support of the *LIN Specification 2.2*
- Support of SAE J2602
- Autobauding; i.e., synchronization using the first received LIN frame
- Programmable baud rate between 1kBit/s and 20kBit/s
- Programmable LIN bus idle time
- Programmable LIN wake up time
- LIN frame header length supervision
- Validation of protected frame identifiers (parity check)
- Bit sample majority (7/16, 8/16, 9/16)
- Rx monitoring
- Rx polarity switch
- Tx bit boundary cancel
- Tx polarity switch
- Unambiguous break field detection to support correct error response reporting
- Suppression of active LIN frame reception (invalid identifier >> wait for next LIN frame)
- AMBA 3 AHB-Lite host controller interface

4.7.1. Functional Description

The "Sync" module synchronizes the incoming serial bit-stream by the ASIC clock. To avoid malfunction due to spikes on the data stream, glitch suppression is implemented by checking for a minimum pulse width of three clock cycles.

The basic data receiver of the serial communication interface is the "UART" module with a byte-filled receiver, triggered by the first falling edge of the start bit and terminated after eight consecutive data bits with the stop bit data value "1." The oversampling rate of each data bit is 16, and the bit data is determined by the bit-sample majority decision of samples 7/16, 8/16 and 9/16.

The "UART" transmitter when triggered by the software allows sending byte fields with the corresponding programmed baud rate. For LIN protocol requirements, the receiver monitoring function is implemented to check the consistency of the bidirectional LIN physical line.

The Break Sync detector is used to identify the beginning of a new LIN frame, and it checks the "Sync" byte field for consistency with the programmed baud rate. The durations of the Break field low phase and Break field delimiter are measured and can be monitored by the software. Additional software-defined thresholds are used for successful frame synchronization. The LIN header inter-byte space between the "Sync" byte field and the Protected Identifier field is measured to evaluate the maximum LIN frame header length.

The "Autobauding" module determines the baud rate on the basis of the first received Break field and Sync field. It ensures that the first received frame is synchronized and valid for LIN communication. The duration of the Break field low phase and Break field delimiter is measured and can be monitored by the software. To support a complete Break Sync field pattern validation, the Sync field's bit 7 data value and the stop bit data value are verified.

© 2016 Integrated Device Technology, Inc.

The "IdleWakeUp" module supports the *LIN Protocol Specification 2.2,* chapters 2.6.2 WAKE UP and 2.6.3 GO TO SLEEP. The threshold for the dominant pulse length after a recessive-to-dominant change for the WAKE UP scenario is defined by a programmable parameter. The inactivity time length for the GO TO SLEEP scenario is defined by a programmable threshold parameter. This allows significant flexibility for low pulse length and inactivity time definitions.

4.7.2. Overview of Registers for LIN ahb Controller

4.7.2.1. Register "LIN_CFG" – Configuration

Table 4.32 Register `LIN_CFG` – system address 4000 1800$_{HEX}$

Name	Bits	Default	Access	Description
rx_enable	[0]	00000000$_{HEX}$	RW	UART receive process 0_{BIN}: disable 1_{BIN}: enable
tx_enable	[1]		RW	UART transmit process 0_{BIN}: disable 1_{BIN}: enable
break_sync	[2]		RW	Break Sync Detection control 0_{BIN}: disable 1_{BIN}: enable
autobauding	[3]		RW	Autobauding process control 0_{BIN}: disable 1_{BIN}: enable
idle	[4]		RW	Idle monitor control 0_{BIN}: disable 1_{BIN}: enable
apply-autobaudrate	[5]		RW	Evaluated autobaudrate is applied to Break Sync and UART process 0_{BIN}: apply `ctrl_baudrate` defined by software (`LIN_BAUDRATE` register) 1_{BIN}: apply `auto_baudrate` (`LIN_BAUDRATE` register)
rx_monitoring	[6]		RW	Receiver monitoring at active transmitting 0_{BIN}: transmitted signal is validated at $T_{BIT}/2$ 1_{BIN}: transmitted serial signal is validated by receiver process
wakeup	[7]		RW	UART receive process 0_{BIN}: off 1_{BIN}: wakeup monitor active

Name	Bits	Default	Access	Description
break_trigger	[8]		W	Strobe Bit: Wait for new Break field reception 0_{BIN}: no action 1_{BIN}: suppress UART receiver flags until new Break Sync field reception
rxdat0_mode	[9]		RW	Rxdat0 mode 0_{BIN}: off 1_{BIN}: if 11 consecutive bits = '0' >> suppress rx status flags assertion 1_{BIN}: if 10 consecutive bits = '0' >> extend rx status flag assertion at 10.5 T_{BIT} 1_{BIN}: if fewer than 10 consecutive bit ='0' >> rx status flag assertion at 9.5 T_{BIT}
rx_start_bit_verification	[10]		RW	Receiver start bit data verification: 0_{BIN}: start bit data not checked 1_{BIN}: start bit data checked (sampling point defined by `rx_bit_sample_majority` bit [12] below)
tx_bit_boundary	[11]		RW	Cancel transmission (linTxD = recessive 1_{BIN}) at bit boundary if framing error or data error detected 0_{BIN}: off 1_{BIN}: active
rx_bit_sample_majority	[12]		RW	Receiver sampling mode 0_{BIN}: sampling time point = $T_{BIT}/2$ 1_{BIN}: sampling time points = 7/16 and 8/16 and 9/16 T_{BIT} Bit samples majority determines the bit data.
idle_bit_sample_majority	[13]		RW	LIN bus idle detector sampling mode 0_{BIN}: continuous monitoring of rx_sync line 1_{BIN}: sampling points = 2^{16} clock period (1.6µs, 3.2µs, 6.4µs, 12.8µs); the majority of 3 consecutive samples determines the bit data.
wakeup_bit_sample_majority	[14]		RW	LIN wakeup detector sampling mode 0_{BIN}: continuous monitoring of rx_sync line 1_{BIN}: sampling points = 2^{16} clock periods (1.6µs, 3.2µs, 6.4µs, 12.8µs); the majority of 3 consecutive samples determines the bit data.
break_sync_bit_sample_majority	[15]		RW	Break Sync detector sampling mode 0_{BIN}: sampling time point $T_{BIT}/2$ 1_{BIN}: sampling time points: 7/16 & 8/16 & 9/16 T_{BIT} Bit samples majority determines the bit data.

Name	Bits	Default	Access	Description
autobauding_threshold	[19:16]		RW	Break low phase length threshold for autobauding process 0000_{BIN}: 11 T_{BIT} 0001_{BIN}: 11 + 11/128 T_{BIT} 0010_{BIN}: 11 + 11/64 T_{BIT} 0011_{BIN}: 11 + 11/32 T_{BIT} 0100_{BIN}: 11 + 11/16 T_{BIT} 1001_{BIN}: 11 – 11/128 T_{BIT} 1010_{BIN}: 11 – 11/64 T_{BIT} 1011_{BIN}: 11 – 11/32 T_{BIT} 1100_{BIN}: 11 – 11/16 T_{BIT} Others: 11 T_{BIT}
rx_start	[20]		W	Test strobe register to trigger receive sequence; read as 0_{BIN} 0_{BIN}: off 1_{BIN}: trigger receive sequencer
rx_inverse	[21]		RW	Test register for inverse of linRxD bit value 0_{BIN}: off 1_{BIN}: linRxD bit value inverted
tx_start	[22]		W	Test strobe register to trigger transmit sequencer; read as 0_{BIN} 0_{BIN}: off 1_{BIN}: trigger transmit sequencer
tx_inverse	[23]		RW	Test register for inversion of linTxD bit value 0_{BIN}: off 1_{BIN}: linTxD bit value inverted
tx_start_stop	[24]		RW	Test register for start and stop bit value for byte field transmission 0_{BIN}: start_bit = 0_{BIN} and stop_bit = 1_{BIN} 1_{BIN}: start_bit = bit [8] in `LIN_TXDATA` register and stop_bit = bit [9] in `LIN_TXDATA` register
unlock	[25]		RW	Test register for basic UART mode 0_{BIN}: Rx and Tx are locked 1_{BIN}: Rx and Tx are unlocked
Unused	[31:26]		RO	Unused; read as 0.

4.7.2.2. Register “LIN_RXDATA” – RX data

Table 4.33 Register LIN_RXDATA – system address 4000 1804$_{HEX}$

Name	Bits	Default	Access	Description
rx_data	[0:7]	00_{HEX}	RO	Received byte field data value.
rx_start_bit	[8]	0_{BIN}	RO	Received byte field start bit data.
rx_stop_bit	[9]	0_{BIN}	RO	Received byte field stop bit data.
rx_sync	[10]	0_{BIN}	RO	linRxD bit data synchronized by clock. Note: can be changed after reset (sampling of serial input stream after reset).
Unused	[31:11]		RO	Unused; read as 0.

4.7.2.3. Register “LIN_TXDATA” – TX data

Table 4.34 Register LIN_TXDATA – system address 4000 1808$_{HEX}$

Name	Bits	Default	Access	Description
tx_data	[7:0]	00_{HEX}	W	Transmit byte field value (write only, read as 0).
tx_start_bit	[8]	0_{BIN}	W	Transmit byte field start bit data (write only, read as 0). Only applicable if unlock = 1_{BIN} and transfer size is word or halfword.
tx_stop_bit	[9]	0_{BIN}	W	Transmit byte field stop bit value (write only, read as 0). Only applicable if unlock = 1_{BIN} and transfer size is word or halfword
Unused	[31:10]		RO	Unused; read as 0.

4.7.2.4. Register “LIN_HEADERLEN” – LIN header length

Table 4.35 Register LIN_HEADERLEN – system address 4000 180C$_{HEX}$

Name	Bits	Default	Access	Description
T_BRKFLD_16	[3:0]	0000_{BIN}	RO	Measured length of Break field low phase (T_{BIT}/16) 0000_{BIN}: 0/16 T_{BIT} 0001_{BIN}: 1/16 T_{BIT} … 1111_{BIN}: 15/16 T_{BIT}
T_BRKFLD	[8:4]	0000_{BIN}	RO	Measured length of Break field low phase (T_{BIT}) 0000_{BIN}: 0 T_{BIT} 0001_{BIN}: 1 T_{BIT} …. 1111_{BIN}: 31 T_{BIT}
Unused	[15:9]		RO	Unused; always read as 0.

Name	Bits	Default	Access	Description
T_BRKDEL_16	[19:16]	0000_{BIN}	RO	Measured length of Break field delimiter (T_{BIT}/16) 0000_{BIN}: 0/16 T_{BIT} 0001_{BIN}: 1/16 T_{BIT} … 1111_{BIN}: 15/16 T_{BIT}
T_BRKDEL	[23:20]	0000_{BIN}	RO	Measured length of Break field low phase (T_{BIT}) 0000_{BIN}: 0 T_{BIT} 0001_{BIN}: 1 T_{BIT} …. 1111_{BIN}: 31 T_{BIT}
rxOverflow_LIN_irq	[4]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
wrCollision_LIN_irq	[5]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
txOff_irq	[6]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
inactive_irq	[7]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
Unused	[31:8]		RO	Unused; read as 0.

4.7.2.5. Register “LIN_BAUDRATE” – LIN baud rate

Table 4.36 Register LIN_BAUDRATE – system address 4000 1810$_{HEX}$

Name	Bits	Default	Access	Description
ctrl_baudrate	[14:0]	$03E8_{HEX}$	RO	**Controller defined baudrate** ctrl_baudrate = Tbit / PI ASIC clock period (50ns) Example: $03E8_{HEX}$ = 50µs / 50ns
Unused	[15]		RO	Unused; always write as 0.
auto_baudrate	[30:16]	0000_{HEX}	RW	**Evaluated baudrate from autobauding process** Tbit = auto_baudrate x PI ASIC clock period (50ns)
Unused	[31]		RO	Unused; read as 0

4.7.2.6. Register "LIN_BRKLOW" – LIN Break field low phase

Table 4.37 Register LIN_BRKLOW – system address 4000 1814$_{HEX}$

Name	Bits	Default	Access	Description
brk_low_min_threshold/16	[3:0]	0000_{BIN}	RW	**Minimum threshold of Break field low phase (T_{BIT}/16)** 0000_{Bin}: 0/16 T_{BIT} 0001_{Bin}: 1/16 T_{BIT} … 1111_{Bin}: 15/16 T_{BIT}
brk_low_min_threshold	[8:4]	01011_{BIN}	RW	**Minimum threshold of Break field low phase (T_{BIT})** 00000_{Bin}: 0 T_{BIT} 00001_{Bin}: 1 T_{BIT} … 11111_{Bin}: 31 T_{BIT}
Unused	[15:9]		RO	Unused; always write as 0.
brk_low_max_threshold/16	[19:16]	1010_{BIN}	RW	**Maximum threshold of Break field low phase (T_{BIT}/16)** 0000_{Bin}: 0/16 T_{BIT} 0001_{Bin}: 1/16 T_{BIT} … 1111_{Bin}: 15/16 T_{BIT}
brk_low_max_threshold	[24:20]	1010_{BIN}	RW	**Maximum threshold of Break field low phase (T_{BIT})** 00000_{Bin}: 0 T_{BIT} 00001_{Bin}: 1 T_{BIT} … 11111_{Bin}: 31 T_{BIT}
Unused	[31:25]		RO	Unused; read as 0.

4.7.2.7. Register "LIN_HINTERBRKDEL" – LIN interbyte and Break field delimiter

Table 4.38 Register LIN_HINTERBRKDEL – system address 4000 1818$_{HEX}$

Name	Bits	Default	Access	Description
brk_delimiter_min_threshold/16	[3:0]	1010_{BIN}	RW	**Minimum threshold of Break field delimiter (T_{BIT}/16)** 0000_{Bin}: 0/16 T_{BIT} 0001_{Bin}: 1/16 T_{BIT} … 1111_{Bin}: 15/16 T_{BIT}
brk_delimiter_min_threshold	[7:4]	0000_{BIN}	RW	**Minimum threshold of Break field delimiter (T_{BIT})** 0000_{Bin}: 0 T_{BIT} 0001_{Bin}: 1 T_{BIT} … 1111_{Bin}: 15 T_{BIT}
brk_delimiter_max_threshold/16	[11:8]	0000_{BIN}	RW	**Maximum threshold of Break field delimiter (T_{BIT}/16)** 0000_{Bin}: 0/16 T_{BIT} 0001_{Bin}: 1/16 T_{BIT} … 1111_{Bin}: 15/16 T_{BIT}
brk_delimiter_max_threshold	[15:12]	1110_{BIN}	RW	**Maximum threshold of Break field delimiter (T_{BIT})** 0000_{Bin}: 0 T_{BIT} 0001_{Bin}: 1 T_{BIT} … 1111_{Bin}: 15 T_{BIT}
h_interbyte_max_threshold/16	[19:16]	0000_{BIN}	RW	**Maximum threshold of header inter-byte space (T_{BIT}/16)** 0000_{Bin}: 0/16 T_{BIT} 0001_{Bin}: 1/16 T_{BIT} … 1111_{Bin}: 15/16 T_{BIT}
h_interbyte_max_threshold	[23:20]	1110_{BIN}	RW	**Maximum threshold of header inter-byte space (T_{BIT})** 0000_{Bin}: 0 T_{BIT} 0001_{Bin}: 1 T_{BIT} … 1111_{Bin}: 15 T_{BIT}
Unused	[31:24]		RO	Unused; read as 0

4.7.2.8. Register "LIN_WAKEUPIDLE" – LIN wakeup threshold and LIN idle

Table 4.39 Register `LIN_WAKEUPIDLE` – system address 4000 181C$_{HEX}$

Name	Bits	Default	Access	Description
bus_idle_th	[15:0]	4C4C$_{HEX}$	RW	**LIN bus idle threshold** (bit [4] in the `LIN_CFG` register = 1$_{BIN}$**)** **Exceeding bus idle threshold >>** bit [5] in the `LIN_STAT` register = 1$_{BIN}$; see Table 4.42 `bus_idle_th` = $T_{busidle}$ / (2^{12} x 50ns) Example: 2^{12} x dec(4C4C$_{HEX}$) x 50ns = 4s = $T_{busidle}$ (See the *LIN Protocol Specification Rev. 2.2* for the definition of $T_{busidle}$.)
wakeup_th	[31:16]	05DC$_{HEX}$	RW	**LIN wakeup threshold** (bit [7] in the `LIN_CFG` register = 1$_{BIN}$**)** **Exceeding wakeup threshold >>** bit [6] in the `LIN_STAT` register = 1$_{BIN}$ `wakeup_th` = T_{wakeup} / (2 x 50ns) See the *LIN Protocol Specification Rev. 2.2* for the definition of T_{wakeup})

4.7.2.9. Register "LIN_IREN" – Interrupt enable

Table 4.40 Register `LIN_IREN` – system address 4000 1820$_{HEX}$

Name	Bits	Reset	Access	Description
rx_byte_field_complete_en	[0]	0$_{BIN}$	RW	**Receive byte field complete interrupt enable** 0$_{BIN}$: bit [0] in the `LIN_STAT` register is disabled from driving interrupt request signal linIrq 1$_{BIN}$: bit [0] in the `LIN_STAT` register drives interrupt request linIrq
tx_byte_field_complete_en	[1]	0$_{BIN}$	RW	**Transmit byte field complete interrupt enable** 0$_{BIN}$: bit [1] in the `LIN_STAT` register is disabled from driving interrupt request signal linIrq 1$_{BIN}$: bit [1] in the `LIN_STAT` register drives interrupt request signal linIrq
brksync_complete_en	[2]	0$_{BIN}$	RW	**Break Sync complete interrupt enable** 0$_{BIN}$: bit [2] in the `LIN_STAT` register is disabled from driving interrupt request signal linIrq 1$_{BIN}$: bit [2] in the `LIN_STAT` register drives interrupt request signal linIrq
auto_sync_complete_en	[3]	0$_{BIN}$	RW	**Autobauding Sync complete interrupt enable** 0$_{BIN}$: bit [3] in the `LIN_STAT` register is disabled from driving interrupt request signal linIrq. 1$_{BIN}$: bit [3] in the `LIN_STAT` register drives interrupt request signal linIrq

Name	Bits	Reset	Access	Description
autobauding_complete_en	[4]	0_{BIN}	RW	**Autobauding complete interrupt enable** 0_{BIN}: bit [4] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [4] in the LIN_STAT register drives interrupt request signal linIrq
bus_idle_en	[5]	0_{BIN}	RW	**Bus idle interrupt enable** 0_{BIN}: bit [5] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [5] in the LIN_STAT register drives interrupt request signal linIrq
wake_up_en	[6]	0_{BIN}	RW	**Wake up interrupt enable** 0_{BIN}: bit [6] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [6] in the LIN_STAT register drives interrupt request signal linIrq
parity_error_en	[7]	0_{BIN}	RW	**PID field parity error interrupt enable** 0_{BIN}: bit [7] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [7] in the LIN_STAT register drives interrupt request signal linIrq
brkfld_error_en	[8]	0_{BIN}	RW	**Break field interrupt enable** 0_{BIN}: bit [8] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [8] in the LIN_STAT register drives interrupt request signal linIrq
brkdel_error_en	[9]	0_{BIN}	RW	**Break delimiter interrupt enable** 0_{BIN}: bit [9] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [9] in the LIN_STAT register drives interrupt request signal linIrq
brksync_error_en	[10]	0_{BIN}	RW	**Break Sync field interrupt enable** 0_{BIN}: bit [10] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [10] in the LIN_STAT register drives interrupt request signal linIrq
h_interbyte_error_en	[11]	0_{BIN}	RW	**Header interbyte space interrupt enable** 0_{BIN}: bit [11] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [11] in the LIN_STAT register drives interrupt request signal linIrq
rx_overrun_en	[12]	0_{BIN}	RW	**Receive buffer overrun interrupt enable** 0_{BIN}: bit [12] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [12] in the LIN_STAT register drives interrupt request signal linIrq
rx_framing_error_en	[13]	0_{BIN}	RW	**Receive framing error interrupt enable** 0_{BIN}: bit [13] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [13] in the LIN_STAT register drives interrupt request signal linIrq

Name	Bits	Reset	Access	Description
rx_data_error_en	[14]	0_{BIN}	RW	**Receive data error interrupt enable** 0_{BIN}: bit [14] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [14] in the LIN_STAT register drives interrupt request signal linIrq
tx_overrun_en	[15]	0_{BIN}	RW	**Transmit buffer overrun interrupt enable** 0_{BIN}: bit [15] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [15] in the LIN_STAT register drives interrupt request signal linIrq
tx_framing_error_en	[16]	0_{BIN}	RW	**Transmit framing error interrupt enable** 0_{BIN}: bit [16] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [16] in the LIN_STAT register drives interrupt request signal linIrq
tx_data_error_en	[17]	0_{BIN}	RW	**Transmit data error interrupt enable** 0_{BIN}: bit [17] in the LIN_STAT register is disabled from driving interrupt request signal linIrq 1_{BIN}: bit [17] in the LIN_STAT register drives interrupt request signal linIrq
Unused	[31:18]		RO	Unused; read as 0.

4.7.2.10. Register “LIN_CLI” – Interrupt clear

Table 4.41 Register LIN_CLI – system address 4000 1824$_{HEX}$

Name	Bits	Reset	Access	Description
rx_byte_field_complete_cl	[0]	0_{BIN}	W	**Receive byte field complete** 0_{BIN}: no action 1_{BIN}: reset bit [0] in the LIN_STAT register
tx_byte_field_complete_cl	[1]	0_{BIN}	W	**Transmit byte field complete** 0_{BIN}: no action 1_{BIN}: reset bit [1] in the LIN_STAT register
brksync_complete_cl	[2]	0_{BIN}	W	**Break Sync complete** 0_{BIN}: no action 1_{BIN}: reset bit [2] in the LIN_STAT register
auto_break_sync_complete_cl	[3]	0_{BIN}	W	**Autobauding Break Sync complete** 0_{BIN}: no action 1_{BIN}: reset bit [3] in the LIN_STAT register
autobauding_complete_cl	[4]	0_{BIN}	W	**Autobauding complete** 0_{BIN}: no action 1_{BIN}: reset bit [4] in the LIN_STAT register
bus_idle_cl	[5]	0_{BIN}	W	**Bus idle** 0_{BIN}: no action 1_{BIN}: reset bit [5] in the LIN_STAT register

Name	Bits	Reset	Access	Description
wake_up_cl	[6]	0_{BIN}	W	**Wake up** 0_{BIN}: no action 1_{BIN}: reset bit [6] in the LIN_STAT register
parity_error_cl	[7]	0_{BIN}	W	**PID field parity error** 0_{BIN}: no action 1_{BIN}: reset bit [7] in the LIN_STAT register
brkfld_error_cl	[8]	0_{BIN}	W	**Break field** 0_{BIN}: no action 1_{BIN}: reset bit [8] in the LIN_STAT register
brkdel_error_cl	[9]	0_{BIN}	W	**Break delimiter** 0_{BIN}: no action 1_{BIN}: reset bit [9] in the LIN_STAT register
brksync_error_cl	[10]	0_{BIN}	W	**Break Sync field** 0_{BIN}: no action 1_{BIN}: reset bit [10] in the LIN_STAT register
h_interbyte_error_cl	[11]	0_{BIN}	W	**Header interbyte space** 0_{BIN}: no action 1_{BIN}: reset bit [11] in the LIN_STAT register
rx_overrun_cl	[12]	0_{BIN}	W	**Receive buffer overrun** 0_{BIN}: no action 1_{BIN}: reset bit [12] in the LIN_STAT register
rx_framing_error_cl	[13]	0_{BIN}	W	**Receive framing error** 0_{BIN}: no action 1_{BIN}: reset bit [13] in the LIN_STAT register
rx_data_error_cl	[14]	0_{BIN}	W	**Receive data error** 0_{BIN}: no action 1_{BIN}: reset bit [14] in the LIN_STAT register
tx_overrun_cl	[15]	0_{BIN}	W	**Transmit buffer overrun** 0_{BIN}: no action 1_{BIN}: reset bit [15] in the LIN_STAT register
tx_framing_error_cl	[16]	0_{BIN}	W	**Transmit framing error** 0_{BIN}: no action 1_{BIN}: reset bit [16] in the LIN_STAT register
tx_data_error_cl	[17]	0_{BIN}	W	**Transmit data error** 0_{BIN}: no action 1_{BIN}: reset bit [17] in the LIN_STAT register
Unused	[31:18]		----	Unused

4.7.2.11. Register "LIN_STAT" – Status

Table 4.42 Register LIN_STAT – system address 4000 1828$_{HEX}$

Name	Bits	Reset	Typ	Description
rx_byte_field_complete	[0]	0_{BIN}	RO	**Receive byte field complete interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
tx_byte_field_complete	[1]	0_{BIN}	RO	**Transmit byte field complete interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
brksync_complete	[2]	0_{BIN}	RO	**Break Sync complete interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
auto_sync_complete	[3]	0_{BIN}	RO	**Autobauding Sync complete interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
autobauding_complete	[4]	0_{BIN}	RO	**Autobauding complete interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
bus_idle	[5]	0_{BIN}	RO	**Bus idle interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
wake_up	[6]	0_{BIN}	RO	**Wake up interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
parity_error	[7]	0_{BIN}	RO	**PID field parity error interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
brkfld_error	[8]	0_{BIN}	RO	**Break field interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
brkdel_error	[9]	0_{BIN}	RO	**Break delimiter interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
brksync_field_error	[10]	0_{BIN}	RO	**Break Sync field interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
h_interbyte_error	[11]	0_{BIN}	RO	**Header interbyte space interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
rx_overrun	[12]	0_{BIN}	RO	**Receive buffer overrun interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
rx_framing_error	[13]	0_{BIN}	RO	**Receive framing error interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request

Name	Bits	Reset	Typ	Description
rx_data_error	[14]	0_{BIN}	RO	**Receive data error interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
tx_overrun	[15]	0_{BIN}	RO	**Transmit buffer overrun interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
tx_framing_error	[16]	0_{BIN}	RO	**Transmit framing error interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
tx_data_error	[17]	0_{BIN}	RO	**Transmit data error interrupt request** 0_{BIN}: idle 1_{BIN}: interrupt request
Unused	[31:18]		RO	Unused; read as 0

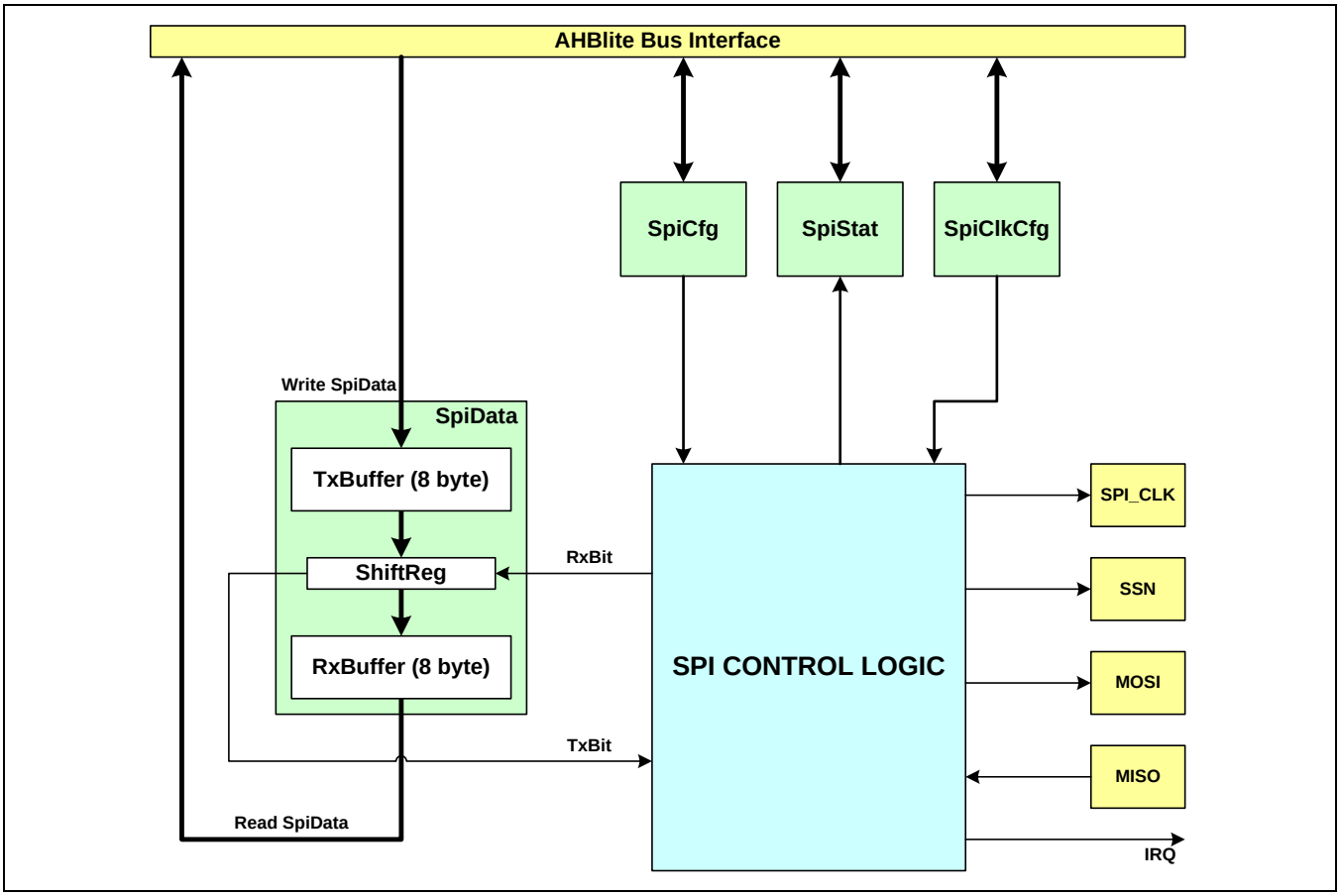
© 2016 Integrated Device Technology, Inc.

4.8. SPIB8

4.8.1. Introduction

The SPIB8 module provides a four-wire SPI master module. The three lines SPI_CLK, MOSI and MISO are fully controlled by hardware while the SSN line must be controlled by software to guarantee the required setup and hold times. In addition to the AHB-Lite bus interface, this SPI module has an active-high interrupt line and an additional input to disable the module. This additional input is needed as the rising edge of SSN will execute the command sent to the SBC. To avoid any problems when the power or system clock is disabled, this input will be triggered by the PMU and goes to deep sleep (i.e., any power-down mode on the SBC; see section 4.3.3).

Figure 4.7 SPIB8 Block Diagram



4.8.2. SPI Signal Description

4.8.2.1. SPI Clock (SPI_CLK)

The SPI clock is used to synchronize the data transfer between the master and the selected slave. The SPI clock is an output of this master and an input to the connected slave (SBC). This module generates the clock if it is enabled and if data is present to be sent. Otherwise the clock line is kept at the configured polarity level.

4.8.2.2. SPI Slave Select (SSN)

The low-active SPI slave select signal is directly driven by bit [14] of the SPICFG_B8 register (see Table 4.43). This means that this line is completely under software control. Software must ensure that the required setup and hold times as well as the required protocol for the SBC are in the defined range.

4.8.2.3. SPI Master Out Slave In (MOSI)

The MOSI pin is used to transfer data from the master to the (selected) slave. Data on this pin is always transferred with the most significant bit (MSB) first.

4.8.2.4. SPI Master In Slave Out (MISO)

The MISO pin is used to transfer data from the slave to the master. Data on this pin is always expected with the most significant bit (MSB) first. It can be selected by software if the MISO line is sampled at the middle (default) or the end of a transmitted bit. The latter case is added to relax the timing when a fast SPI clock is selected.

4.8.3. Functional Description

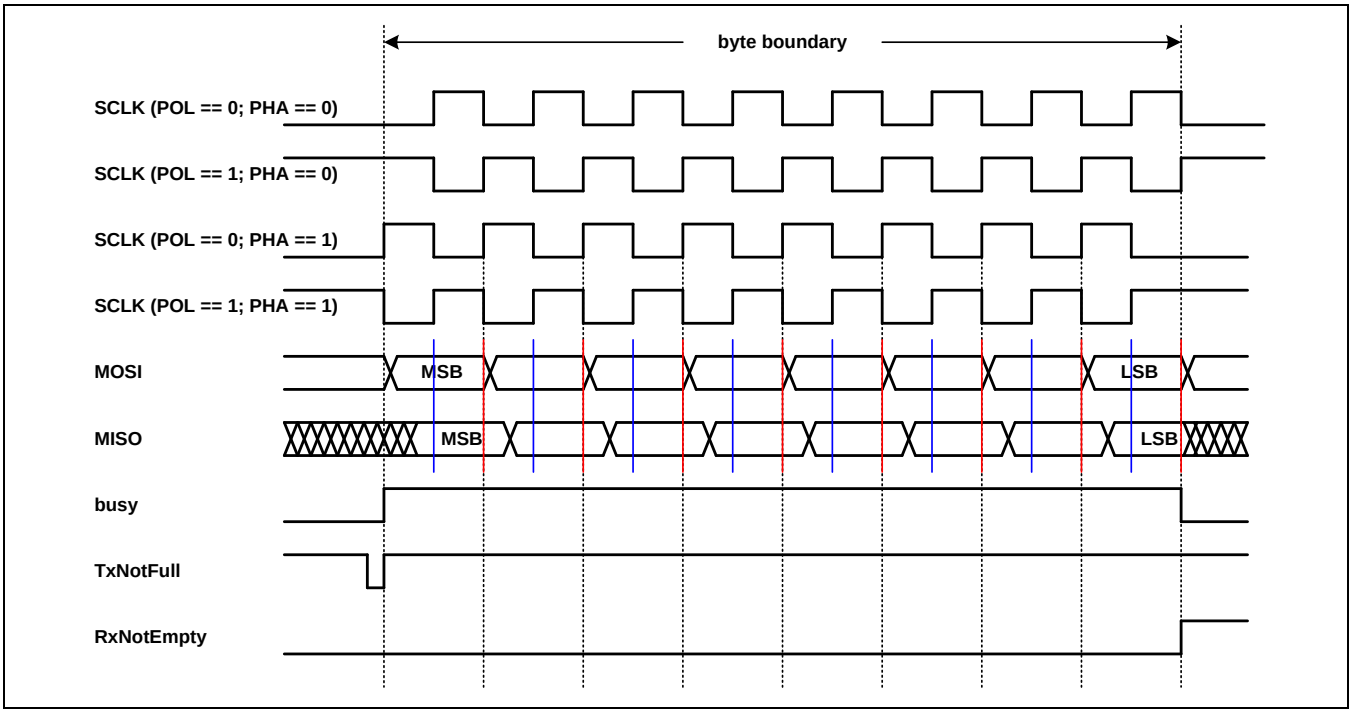
The SPIB8 master initiates all transfers on the SPI bus. To start a transfer, the module must be enabled (set the SpiEn bit [15] in the SPICFG_B8 register to 1_{BIN}; see Table 4.43), the slave must be activated (set SSN to 0), and the required clock behavior must be configured via the CDIV_B8, CPHA_B8 and CPOL_B8 bits in the SPICLKCFG_B8 register; see Table 4.44). The required interrupt sources must also be enabled. Note that the TX buffer is empty at the beginning.

When the required setup time for the SSN line has expired, place at least the first byte to be transmitted into the TX buffer. This also clears the TxNotFull flag bit [2] in the SPISTAT_B8 register. In the next system clock cycle, this byte is transferred into the shift register, the clock line is driven appropriately, and the TxNotFull flag (when only one byte was written into the TX buffer) and the busy flag (bit [7] in the SPISTAT_B8 register) are set. The first byte is shifted out of the MOSI line and the SPI clock is generated as configured.

Simultaneously the MISO line is sampled and shifted in. Normally, the MISO line is sampled in the middle of a transmitted bit (blue lines in Figure 4.8). Although the MOSI line changes its value at the same time as the SPI clock, there is a delay regarding the MISO line as first the clock must be driven out of the chip into the connected slave and then the data must be driven back from the connected slave. To relax the timing, especially for fast SPI clocks, the `SamplePos` bit [13] in the `SPICLKCFG_B8` register can be configured so that the RX data is sampled at the end of a transmitted bit (red lines in Figure 4.8). If the complete byte is shifted in and the RX buffer is not full, the byte is stored into the RX buffer at the byte boundary and the `RxNotEmpty` flag (bit [1] in the `SPISTAT_B8` register) is set, signaling the end of the byte transfer. In the case that the RX buffer is already full and no byte is read from RX buffer in the same cycle, the byte currently received is rejected (lost) and the `RxOf` flag bit [0] in the `SPICFG_B8` register is set.

Because the SPI module operates in a full-duplex mode, a dummy byte must be placed into TX buffer if only a byte has to be read from the slave.

Figure 4.8 SPI Bus and Status Flags for a Single Byte Transfer



When the user writes a new byte into TX buffer before the end of the byte transfer, the transfer for the new byte starts immediately after the actual transfer. This means that the busy flag stays active at the end of the first transmitted byte.

As it can be possible that the software disables the SPI while a transfer is in progress (not recommended), bytes could be present inside the TX and/or RX buffer indicated by the values `NoTxFree` or `NoRxBytes` (bits [23:20] or bits [19:16] respectively in register `SPISTAT_B8`). These bytes can be removed from the buffers by writing a 1 to the `ClrTxBuf` bit [31] or `ClrRxBuffer` bit [30] respectively in the `SPISTAT_B8` register.

As noted above, the user can configure the SPI clock frequency via the CDIV bit field. The SPI clock frequency is calculated using the following equation:

SPI clock frequency = system clock frequency / (2 * (CDIV + 1))

4.8.4. Interrupts and Status Flags

There are eight status flags in this module; seven of them can be enabled to drive the interrupt line. Four of them correspond to the status of the TX or RX buffer: RxNotEmpty, TxNotFull, RxLvl, and TxLvl. They are cleared when data is written to or read from the corresponding buffer. The other three flags that can drive the interrupt line are signaling error conditions: RxOf, WrColl, and RdErr. These three flags are cleared by read access to the status register. The last status bit, Busy, is a read-only signal which reflects the status of the module and is fully controlled by hardware. The other flags are controlled by hardware and by software.

- RxOf: This overflow bit is set by hardware when it is unable to store a received byte into RX buffer (RX buffer is already full). It is cleared by software read access to the status register. To avoid losing information, the set condition has higher priority than the clear condition. This bit is not set when a byte in the RX buffer is read in the same system clock cycle when the received byte will be stored.
- RxNotEmpty: This bit is set by hardware when a received byte is stored into the RX buffer and cleared when all bytes are read by the software. To avoid losing any information, the set condition has higher priority than the clear condition. This situation occurs when the byte in the RX buffer is read in the same system clock cycle when the next received byte will be stored.
- TxNotFull: This flag is active on default. It is cleared when software writes 8 bytes to the TX buffer and is set by hardware when it moves one byte into the shift register. As it might be possible that both actions happen in the same system clock cycle, the clear condition has the higher priority.
- WrColl: This write collision flag is set when the software writes more bytes to TX buffer than free places exist. When one byte is moved into the shift register in the same system clock cycle, its place is also interpreted as free. The bytes software wants to write are completely rejected to avoid loss of data. It is cleared by software read access to the status register.
- RdErr: This read error flag is set when software tries to read more bytes than are present in the RX buffer. It is cleared by software read access to the status register.
- RxLvl: This bit is set by hardware when the number of bytes present in the RX buffer (the NoRxBytes bit field [19:16] in the SPISTAT_B8 register) reaches the level defined by the RxTrigLvl bit field[19:16] in the SPICFG_B8 register. It is cleared by hardware when the number of bytes present in the RX buffer drops below the defined level due to reconfiguration of the level, RX buffer read accesses, or clearing it via ClrRxBuf bit in the SPISTAT_B8 register.
- TxLvl: This bit is set by hardware when the number of free byte locations in the TX buffer (the NoTxFree bit field in the SPISTAT_B8 register) reaches the level defined by the txTrigLvl [23:20] bit field in the SPICFG_B8 register. It is cleared by hardware when the number of free byte locations in the TX buffer drops below the defined level due to reconfiguration of the level or TX buffer write accesses.
- Busy: This bit reflects the status of the SPI module.

4.8.5. Overview of Registers for SPIB8

4.8.5.1. Register "SPICFG_B8" – SPIB8 configuration

Table 4.43 *Register SPICFG_B8 – system address 4000_2000$_{HEX}$; local address is 00$_{HEX}$*

Name	Bits	Reset	Type	Description
RxOf_en	[0]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
RxNotEmpty_en	[1]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
TxNotFull_en	[2]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
WrColl_en	[3]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
RdErr_en	[4]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
RxLvl_en	[5]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
TxLvl_en	[6]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
unused	[12:7]		RO	Unused; always read as 0.
SamplePos	[13]	0_{BIN}	RW	Selects whether data on MISO is sampled at the sampling edge (set to 0) or at shift edge (set to 1). **Note:** Change this bit only when module is disabled (SpiEn == 0; see below) or when no transfer is in progress.
SSN	[14]	1_{BIN}	RW	This bit directly controls the SSN line.
SpiEn	[15]	0_{BIN}	RW	Enable for SPI module.
RxTrigLvl	[19:16]	0100_{BIN}	RW	Defines the number of bytes that must be present in the RX FIFO buffer to activate the RxLvl interrupt
TxTrigLvl	[23:20]	0100_{BIN}	RW	Defines the number of empty locations (bytes) that must be present in TX FIFO buffer to activate the TxLvl interrupt.
Unused	[31:24]		RO	Unused; always read as 0.

4.8.5.2. Register "SPICLKCFG_B8" – SPIB8 clock configuration

***Table 4.44** Register `SPICLKCFG_B8` – system address 4000_2004$_{HEX}$; local address is 08$_{HEX}$*

Name	Bits	Reset	Access	Description
CPOL_B8	[0]	1_{BIN}	RW	Clock polarity; the content of this bit directly reflects the idle state of the clock. **Note:** change this bit only when module is disabled (spien == 0)
CPHA_B8	[1]	1_{BIN}	RW	Clock phase; data is centered to the first (set to 0) or to the second (set to 1) clock edge. **Note:** change this bit only when module is disabled (spien == 0).
CDIV_B8	[7:2]	00001_{BIN}	RW	Clock divider value; spi clock period is 2*(CDIV+1) times the system clock. **Note:** change this bit only when module is disabled (spien == 0) or when no transfer is in progress.
Unused	[31:8]		RO	Unused; read as 0.

4.8.5.3. Register "SPISTAT_B8" – SPIB8 status

Table 4.45 Register SPISTAT_B8 – system address 4000_2008$_{HEX}$; local address is 04$_{HEX}$

Name	Bits	Reset	Access	Description
RxOf	[0]	0_{BIN}	RC	Signals that an Rx overflow has occurred. **Note:** This bit is cleared when its status is read. **Note:** The received byte causing the overflow is rejected; the previous received bit is kept in the RxBuffer.
RxNotEmpty	[1]	0_{BIN}	RO	This bit reflects the status of the RxBuffer. It is set when a new byte is transfered into the empty RxBuffer. It is cleared when SPIDATA is completely read (see Table 4.46).
TxNotFull	[2]	1_{BIN}	RO	This bit reflects the status of the TxBuffer. It is set when at least one byte can be placed into the TxBuffer. It is cleared when no byte can be placed into the TxBuffer.
WrColl	[3]	0_{BIN}	RC	This bit is set when SPIDATA is written while the TxBuffer is already full. **Note:** This bit is cleared when its status is read. **Note:** The bytes to be written causing this error are rejected.
RdErr	[4]	0_{BIN}	RC	This bit is set when SPIDATA is read with more bytes than present in the RxBuffer. **Note:** This bit is cleared when its status is read.
RxLvl	[5]	0_{BIN}	RO	This bit is fully controlled by hardware. It is set when `NoRxBytes` (see bits [19:16] below) has reached the level configured by the `RxTrigLvl` bit field [19:16] in the `SPICFG_B8` register. It is cleared when the RxBuffer is cleared or when enough bytes are read from the RxBuffer so that `NoRxBytes` drops below `RxTrigLvl`.
TxLvl	[6]	1_{BIN}	RO	This bit is fully controlled by hardware. It is set when `NoTxFree` (see bits [23:20] below) has reached the level configured by the `TxTrigLvl` bit field [23:20] in the `SPICFG_B8` register. It is cleared when the TxBuffer is cleared or when enough bytes are written to the TxBuffer so that `NoTxFree` drops below `TxTrigLvl`.
Busy	[7]	0_{BIN}	RO	This bit reflects the status of the SPI module.
Unused	[15:8]		RO	Unused; read as 0.
NoRxBytes	[19:16]	0000_{BIN}	RO	Number of bytes present in the RxBuffer.

Name	Bits	Reset	Access	Description
NoTxFree	[23:20]	1000_{BIN}	RO	Number of free locations (bytes) present in the TxBuffer.
Unused	[29:24]		RO	Unused; read as 0.
ClrRxBuf	[30]	0_{BIN}	WO	Writing a 1 to this bit clears the RxBuffer (strobe register). **Note:** Write only when spi is disabled; always read as 0.
ClrTxBuf	[31]	0_{BIN}	WO	Writing a 1 to this bit clears the TxBuffer (strobe register) **Note:** Write only when spi is disabled; always read as 0.

4.8.5.4. Registers "SPIDATA*_B8" – SPIB8 data

Table 4.46 Accessing the FIFO Buffers – system address 4000_XXXX$_{HEX}$

Name	Address	XXXX	Write Access	Read Access
SPIDATA4B_B8	$0C_{HEX}$	200C	All 4 bytes from data bus are placed into the TxBuffer with lowest byte first	4 bytes are taken from the RxBuffer and placed onto the data bus with the first byte out as the LSB. **Note:** Only word access is possible.
SPIDATA1BU_B8	10_{HEX}	2010	Lowest byte from data bus is placed into the TxBuffer	**Unsigned read:** 1 byte is taken from the RxBuffer and placed onto the data bus (LSB). The upper 3 bytes on the data bus are set to 0. **Note:** Word, halfword, and byte accesses are possible.
SPIDATA1BS_B8	14_{HEX}	2014		**Signed read:** 1 byte is taken from the RxBuffer. This byte is sign extended to form the word sent on the data bus. **Note:** Word, halfword, and byte accesses are possible.
SPIDATA2BU_B8	18_{HEX}	2018	Lowest 2 bytes from data bus are placed into the TxBuffer with lowest byte first	**Unsigned read:** 2 bytes are taken from the RxBuffer and combined into a 16-bit value with the first byte out as LSB. The upper 2 bytes on data bus are set to 0. **Note:** Only word and halfword accesses are possible.
SPIDATA2BS_B8	$1C_{HEX}$	201C		**Signed read:** 2 bytes are taken from the RxBuffer and combined into a 16-bit value with first byte out as LSB. This value is sign extended to form the word sent on the data bus. **Note:** Only word and halfword accesses are possible.
SPIDATA3BU_B8	20_{HEX}	2020	Lowest 3 bytes from data bus are placed into the TxBuffer with lowest byte first	**Unsigned read:** 3 bytes are taken from the RxBuffer and combined into a 24-bit value with the first byte out as the LSB. The highest byte on the data bus is set to 0. **Note:** Only word accesses are possible.
SPIDATA3BS_B8	24_{HEX}	2024		**Signed read:** 3 bytes are taken from the RxBuffer and combined into a 24-bit value with first byte out as the LSB. This value is sign extended to form the word sent on the data bus. **Note:** Only word accesses are possible.

4.9. SPI in ZSYSTEM2

The two SPIs contained in the ZSSC1956 are identical four-wire master interfaces: the SPIB8 is used for communication with the SBC and the other SPI block contained in ZSYSTEM2 can be mapped onto the GPIO pads for external communication. The SPI for external communication is described here. The three lines SCLK, MOSI, and MISO are fully controlled by hardware while the CSN line must be controlled by software.

By default, SCLK is high (CPOL == 1; see Table 4.49). However, it can be changed as needed for the SPI mapped onto the GPIO pads. The SPI clock frequency is configurable by software with a maximum frequency of half of the system clock frequency.

By default, the MISO line is sampled on the second edge of SCLK. However, it can be changed as needed for the SPI mapped onto the GPIO pads as all four possible SPI modes are implemented.

Important: Before the SPI in ZSYSTEM2 can be used, the clock of ZSYSTEM2 must be enabled via register SYS_CLKCFG (see Table 4.5). After enabling the clock for the ZSYSTEM2, the SPI lines must be mapped onto appropriate GPIO pads (see section 4.3.4).

4.9.1. Data Transfers

The SPI master initiates all transfers on the SPI bus. To start a transfer, the module must be enabled (set bit `SpiEn` in the SPI configuration register to 1; see Table 4.47), the required clock behavior must be configured (set `CDIV`, `CPHA` and `CPOL` as needed; see Table 4.49), and the slave must be activated (set `CSN` to 0 in the SPI configuration register). Additionally, the required interrupt sources must be enabled.

Note: The TX buffer is empty at the beginning.

When the required setup time for the CSN line has expired (50ns), the first byte to be transmitted must be placed into the TX buffer. This write access to the TX buffer also clears the `TxEmpty_SPI` flag. In the next system clock cycle, this byte is transferred into the shift register, the clock line is driven appropriately, and the `TxEmpty_SPI` and `Busy` flag are set to 1, which allows the user to place a second byte into the TX buffer. The first byte is shifted out of the MOSI line, and the SPI clock is generated as configured.

Simultaneously, the MISO line is sampled and shifted in. Normally, the MISO line is sampled in the middle of a transmitted bit (blue lines in Figure 4.9). While the MOSI line changes its value at the same time as the SPI clock, there is a delay regarding the MISO line as first the clock must be driven out of the chip into the connected slave and then the data must be driven back from the connected slave. To relax the timing, especially for fast SPI clocks, the `SamplePos` bit in the SPI configuration register (see Table 4.47) can be set so that the RX data is sampled at the end of a transmitted bit (red lines in Figure 4.9).

When the complete byte is shifted in and the RX buffer is empty, the byte is stored into the RX buffer at the byte boundary and the `RxFull_SPI` flag is set to 1 in the SPI status register (see Table 4.50), signaling the end of the byte transfer. If the RX buffer is already full and the byte in the RX buffer is not read in the same cycle, the byte currently received is rejected (lost) and the `RxOverflow_SPI` flag is set to 1 in the SPI status register.

Note: As the SPI module operates in a full-duplex mode, a dummy byte must be placed into the TX buffer if a byte must be read from the slave without any write to the slave.

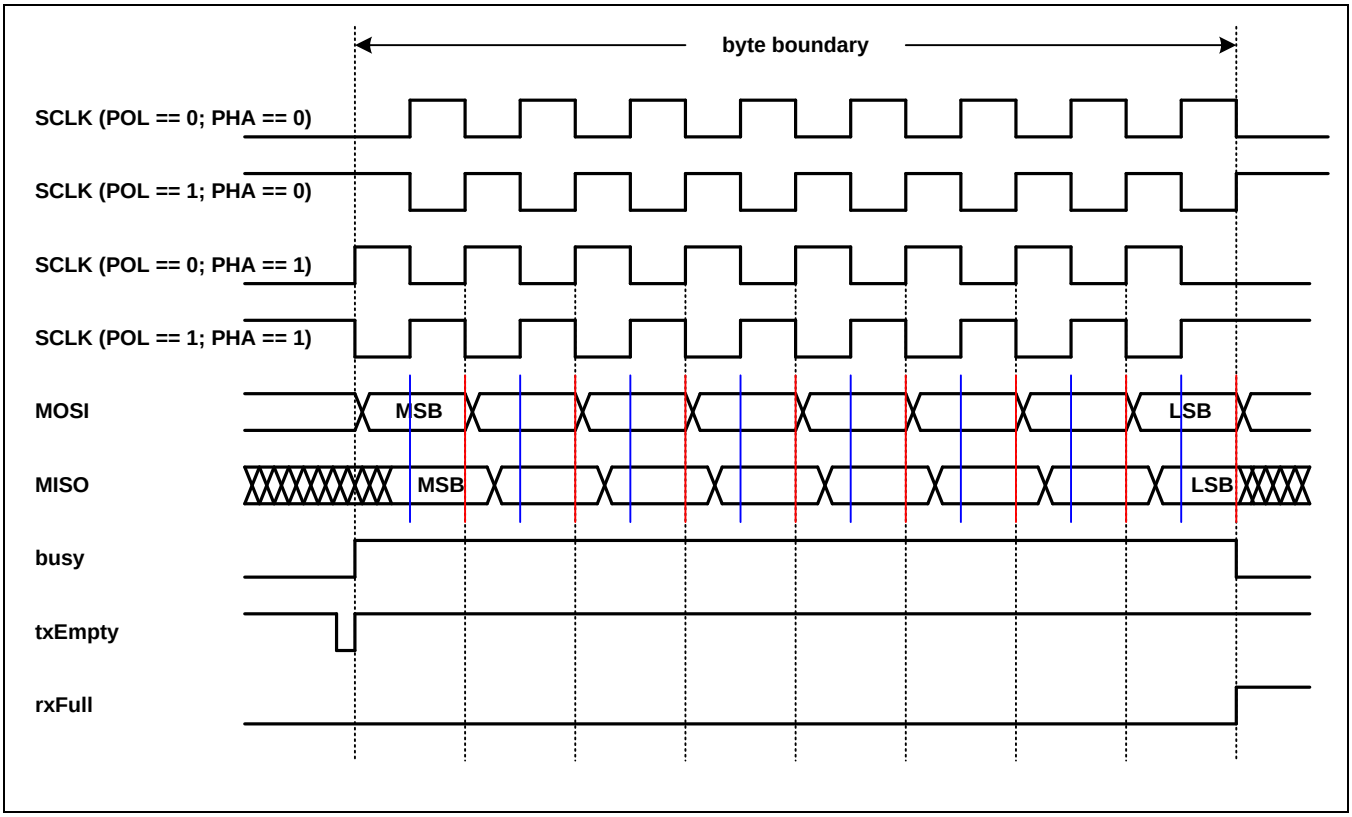
When a new byte is written into the TX buffer before the end of an active byte transfer, the transfer of the new byte starts immediately after the actual transfer. This means that the busy flag stays active at the end of the first transmitted byte.

As it can be possible that the software disables the SPI while a transfer is in progress (not recommended), a byte could be present in the TX buffer indicated by a low value of the `TxEmpty_SPI` flag in the SPI status register. This byte can be removed from the TX buffer by writing a 1 to the `ClrTxBuf` bit of the SPI status register (see Table 4.50); otherwise it would be transmitted when SPI is enabled again.

As mentioned above, the user can configure the SPI clock frequency by the `CDIV` field. The SPI clock frequency is given by the following equation:

SPI clock frequency = system clock frequency / (2 ∗ (CDIV + 1))

Figure 4.9 SPI Bus and Status Flags for a Single Byte Transfer



4.9.2. Interrupts and Status Flags

There are five status flags for this SPI module (see Table 4.50); four of them can be enabled to drive the interrupt line. Two of the flags correspond to the status of the TX or RX buffer. They are cleared when data is written to or read from the corresponding buffer. The other two flags can drive the interrupt line are signaling error conditions. These two flags are cleared by read access to the status register. The fifth status bit is a read only signal which reflects the status of the module and is fully controlled by hardware. The other flags are controlled by hardware and by software:

- `RxOverflow_SPI`: This bit is set to 1 by hardware when it is not able to store a received byte into the RX buffer (RX buffer is already full). It is cleared by software read access to the status register. To prevent losing any information, the set condition has higher priority than the clear condition. This bit is not set when the byte in the RX buffer is read in the same system clock cycle when the next received byte will be stored.
- `RxFull_SPI`: This bit is set to 1 by hardware when a received byte is stored in the RX buffer and cleared when the byte is read by the software. To prevent losing any information, the set condition has higher priority than the clear condition. This situation occurs when the byte in the RX buffer is read in the same system clock cycle when the next received byte will be stored.
- `TxEmpty_SPI`: This flag is active on default (1). It is cleared when software writes a byte to the TX buffer and is set to 1 by hardware when it moves this byte into the shift register. As it might be possible that both actions happen in the same system clock cycle, the clear condition has the higher priority.
- `WrCollision_SPI`: This flag is set when the software writes a byte into the TX buffer while the TX buffer is not empty and its content is not moved into the shift register in the same system clock cycle. The byte that the software attempted to write is rejected to avoid loss of data. It is cleared by software read access to the status register.

4.9.3. Example of SPI Transfer Handling

The following gives a short example of a transfer of six bytes on the SPI bus (SPI2 module):

- Write the SPICLKCFG register (see Table 4.49) with the required `CPOL`, `CPHA` and `CDIV` value.
- Write the SPICFG register with `SpiEn` set to 1 and `CSN` set to 0 and enable the `RxFull_SPI_irq` and `RxOverflow_SPI_irq` interrupts (see Table 4.47). Enabling interrupts can also happen before enabling SPI.
- Write the first TX byte to the TX buffer when the CSN setup time has expired.
- Write the second TX byte to the TX buffer.
- Wait for the `RxFull_SPI` interrupt.
- Read the first RX byte from the RX buffer.
- Write the third TX byte to the TX buffer.
- Wait for `RxFull_SPI` interrupt.
- Read the second RX byte from the RX buffer.
- Write the fourth TX byte to the TX buffer.
- Wait for the `RxFull_SPI` interrupt.
- Read the third RX byte from the RX buffer.
- Write the fifth TX byte to the TX buffer.
- Wait for the `RxFull_SPI` interrupt.
- Read the fourth RX byte from the RX buffer.
- Write the sixth TX byte to the TX buffer.

- Wait for the RxFull_SPI interrupt.
- Read the fifth RX byte from the RX buffer.
- Wait for the RxFull_SPI interrupt.
- Read the sixth RX byte from the RX buffer.
- Write the SPICFG register with CSN set to 1 (and SpiEn set to 0) when the CSN hold time has expired.

When the software is not able to read the RX byte before the next byte is received (RX overflow), the timing can be relaxed by waiting for the first RxFull_SPI interrupt before writing the second TX byte. Instead the second TX byte can be written after this interrupt. This guarantees that no RX overflow can occur but introduces some delay between two consecutive bytes as the module waits for the next byte transfer until data is present.

Note: A special case for releasing CSN is when a power-down command has been sent to SBC. As the SBC will at least disable the clock for the MCU when CSN is released, disabling CSN by software can lead to unwanted behavior. Therefore a hardware mechanism is implemented that disables SPI and releases CSN when a WFI command with the SLEEPDEEP bit set to 1 is executed (see section 4.3.3).

© 2016 Integrated Device Technology, Inc.

4.9.4. Register Overview of SPI2

The register description below is for the SPI2. The SPIB8 register description can be found in the SPIB8 section.

4.9.4.1. Register "Z2_SPICFG" – SPI Configuration

Table 4.47 Register Z2_SPICFG – system address 4000 1C00$_{HEX}$

Name	Bits	Default	Access	Description
RxOverflow_SPI_irq	[0]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
RxFull_SPI_irq	[1]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
TxEmpty_SPI_irq	[2]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
WrCollision_SPI_irq	[3]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
Unused	[4]	0_{BIN}	RO	Unused; always write as 0.
SamplePos	[5]	0_{BIN}	RW	This bit selects whether data on MISO is sampled at the sampling edge (set to 0) or at the shift edge (set to 1). **Note:** Change this bit only when the module is disabled (`SpiEn` == 0) or when no transfer is in progress.
CSN	[6]	1_{BIN}	RW	This bit directly controls the CSN line.
SpiEn	[7]	0_{BIN}	RW	Enable for the SPI module.
Unused	[31:8]		RO	Unused; read as 0

4.9.4.2. Register "Z2_SPIDATA" – SPI Data Buffers

Table 4.48 Register Z2_SPIDATA – system address 4000 1C04$_{HEX}$

Name	Bits	Default	Access	Description
SpiData	[7:0]	00_{HEX}	RW	When writing a byte to this register, the value is stored in the TX buffer. A write access to this register also clears the `TxEmpty_SPI` flag in the status register. When reading this register, the contents of the RX buffer is returned. A read access to this register also clears the `RxFull_SPI` flag in the status register. **Note:** When writing to this register when the TX buffer is full, the TX buffer keeps its content and the written byte is rejected. This is signaled by the `WrCollision_SPI` flag in the status register.
Unused	[31:8]		RO	Unused; read as 0

4.9.4.3. Register "Z2_SPICLKCFG" – SPI Clock Configuration

Table 4.49 Register Z2_SPICLKCFG – system address 4000 1C08$_{HEX}$

Name	Bits	Default	Access	Description
CPOL	[0]	1_{BIN}	RW	Clock polarity; the content of this bit directly reflects the idle state of the SPI clock. **Note:** Change this bit only when the module is disabled (`SpiEn` == 0).
CPHA	[1]	1_{BIN}	RW	Clock phase; data is centered to the first (set to 0) or to the second (set to 1) clock edge. **Note:** Change this bit only when the module is disabled (`SpiEn` == 0).
CDIV	[7:2]	000001_{BIN}	RW	Clock divider value; SPI clock period is 2*(CDIV+1) times the system clock. **Note:** Change this bit only when the module is disabled (`SpiEn` == 0) or when no transfer is in progress.
Unused	[31:8]		RO	Unused; read as 0

4.9.4.4. Register " Z2_SPISTAT" – SPI Status

Table 4.50 Register Z2_SPISTAT – system address 4000 1C0C$_{HEX}$

Name	Bits	Default	Access	Description
RxOverflow_SPI	[0]	0_{BIN}	RC	This bit signals that an Rx overflow occurred. **Note:** This bit is cleared when status is read. **Note:** the received byte causing the overflow is rejected; the previous received bit is kept in the RX buffer.
RxFull_SPI	[1]	0_{BIN}	RO	This bit reflects the status of the RX buffer. It is set when a new byte is transferred into the RX buffer. **Note:** This bit is cleared when `SpiData` is read.
TxEmpty_SPI	[2]	1_{BIN}	RO	This bit reflects the status of the TX buffer. It is set when a byte is transferred from the TX buffer into the shift register. **Note:** This bit is cleared when `SpiData` is written.
WrCollision_SPI	[3]	0_{BIN}	RC	This bit is set when `SpiData` is written while TX buffer is already full. **Note:** This bit is cleared when the status is read.
Busy	[4]	0_{BIN}	RO	This bit reflects the status of the SPI module.
Unused	[6:5]	00_{BIN}	RO	Unused; always write as 0.
ClrTxBuf	[7]	0_{BIN}	WO	Writing a 1 to this bit clears the TX buffer. **Note:** Write only when SPI is disabled; always read as 0.
Unused	[31:8]		RO	Unused; read as 0

4.10. I²C™ in ZSYSTEM2

The master-slave I²C™ module provides an interface to the I²C™ bus, which is compliant to the Philips I²C™ bus specification. It supports all transfer modes (RX and TX; master and slave) and can be connected to busses operating as a slave, single-master, or as one of many masters. It supports true multi-master operation including collision detection and bus arbitration. The 10-bit addressing mode and the high-speed mode are not supported. The maximum possible frequency on the bus is one sixteenth of the internal clock.

Each transfer on the I²C™ bus is controlled by interrupts when software interaction is needed. All registers of this I²C™ module must only be accessed when the device is disabled or when an interrupt is active.

Important: Before the I²C™ in ZSYSTEM2 can be used, the clock of ZSYSTEM2 must be enabled via register SYS_CLKCFG (see Table 4.5). After the clock for the ZSYSTEM2 is enabled, the I²C™ lines must be mapped onto appropriate GPIO pads (see section 4.3.4).

The I²C™ module in ZSYSTEM2 has an active-high interrupt line connected to ARM® interrupt 8.

4.10.1. External Signal Lines

The I²C™ bus consists of two external signal lines, SCL and SDA, for communication between all devices connected to the bus. As SCL is always driven by the master and SDA by various devices, both output drivers operate as open-drain drivers independent of the corresponding bit in the ppNod bit field in the register SYS_MEMPORTCFG.

The I²C™ clock is used to synchronize the data transfer between the devices. During each transfer, the clock is generated by the master on the SCL line, but its low phase can be extended by each connected slave. In slave mode, the device extends the low phase of the ninth bit (ACK) to enable the software to setup the next byte transfer. The incoming clock is synchronized and filtered for resistance against short spikes on the clock line.

The I²C™ data line is always driven by the transmitter. For the first byte of a transfer, the transmitter is always the master transferring the address and the direction of the next bytes. When a transmitter sends a 1 but detects a 0 on the bus, it immediately releases the bus. The incoming data line is synchronized and filtered for resistance against short spikes on the data line.

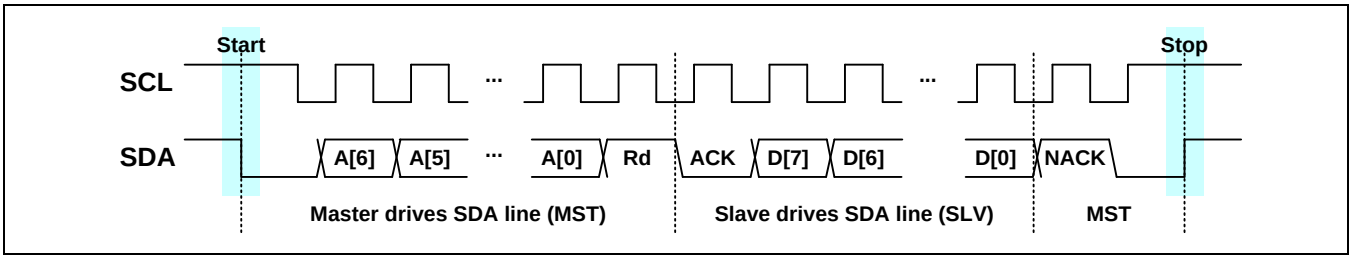
4.10.2. The I²C™ Bus

Each transfer on the I²C™ bus is a read or write access by a master to a slave. All transfers are initiated by a master generating a START condition on a bus followed by the address byte, which must be acknowledged by the addressed slave. Depending on the access type (read; write), data bytes are sent over the bus by the transmitter. Each byte sent on the bus must be 8 bits long followed by an acknowledge bit returned by the receiver. The transfer is terminated by the master generating a STOP condition on the bus or by starting a new transfer by generating a RESTART condition. All bytes are transferred MSB first.

Sequence when the master requests data from the slave:

- Master generates a START condition when the bus is free.
- Master sends the address byte; the slave sends the acknowledge bit in response.
- Slave sends the data bytes; the master sends the acknowledge bit in response to each byte (last byte is NACKed to signal to the slave to release the bus).
- Master generates the STOP condition to release the bus or the RESTART condition to start a new transfer.

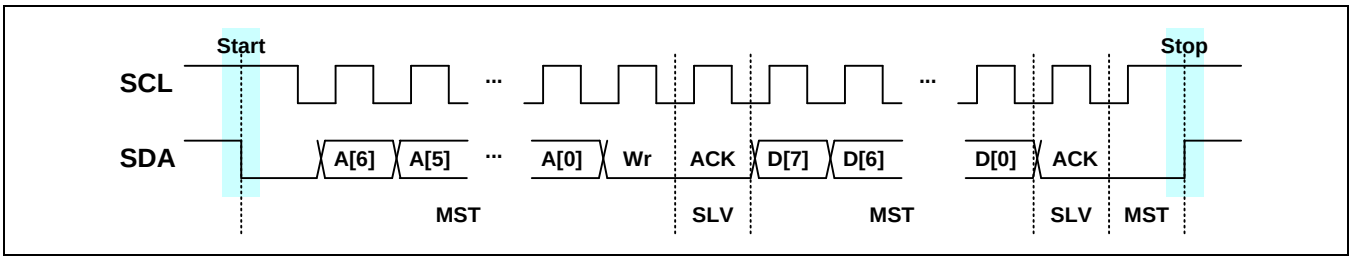
Figure 4.10 Read Transfer Example



Sequence when the master sends data to the slave:

- Master generates a START condition when the bus is free.
- Master sends the address byte; the slave sends the acknowledge bit in response.
- Master sends the data bytes; the slave sends the acknowledge bit in response to each byte (master continues until NACK is received or no more data is to be sent).
- Master generates the STOP condition to release the bus or the RESTART condition to start a new transfer.

Figure 4.11 Write Transfer Example



4.10.3. Bus Conflicts

Bus conflicts can occur when two devices drive the data line SDA at the same time. This can happen when two master devices have generated a START condition at the same time, when two slave devices have the same slave address, or when two slave devices are accessed for writing by the global call address. As the outputs are open-drain drivers, a conflict can be detected by the device driving a 1 as this 1 is overridden by the device driving a 0 on the bus. All these conflicts are handled in hardware.

Exceptions: The following possible conflicts (masters only) cannot be handled in hardware:

- Data bit ←→ STOP
- Data bit ←→ RESTART
- RESTART ←→ STOP

The occurrence of these conflicts must be avoided by software protocols.

Note: The first two conflicts can be avoided if all masters access the same slave with the same amount of bytes. However, different numbers of bytes can be used for different slaves.

Note: The third conflict can be avoided if all masters behave in the same manner, either generating a STOP or a RESTART.

© 2016 Integrated Device Technology, Inc.

Note: Another solution for avoiding these conflicts can be that each master performs a write access without any data to all the other masters to ensure that it is the only master on the bus.

When the ZSSC1956 IBS is accessed as slave, it automatically handles the following conflicts:

- Conflict during transmission of the acknowledge bit following the address byte: this can only occur when another device has the same slave address or the global call address is used and when this device is not ready to be accessed and therefore is sending a NACK. It just releases the bus and ignores all actions on the bus until it detects a RESTART, START, or STOP condition.
- Conflict during transmission of the acknowledge bit following a data byte: this can only occur for a write transfer when another device has the same slave address or the global call address is used and when this device is not ready to receive more data and therefore is sending a NACK. It just releases the bus and ignores all actions on the bus until it detects a RESTART, START, or STOP condition.
- Conflict during transmission of a data byte: this can only occur for a read transfer when another device has the same slave address and both have sent an ACK in response to the address byte. This conflict occurs when this device is transmitting a 1 (recessive value) while the other slave is transmitting a 0 (dominant value). This device immediately releases the bus and generates an interrupt with status S_I2cStConflict (see page 193).

4.10.4. Operating as Slave-Only

When the ZSSC1956 IBS is operating as a pure slave on the I²C™ bus without using the master functionality, the registers `Z2_I2CCLKRATE` and `Z2_I2CCLKRATE2` are not needed as the clock on the I²C™ bus is always generated by the master device. The `stop` and `start` bits of register `Z2_I2CCTRL` (see Table 4.54) must always be set to 0 as the START and STOP conditions are always generated by the master while the `multi` bit must be set to 1 to avoid conflicts when enabling the module (see section 4.10.8).

For the ZSSC1956 IBS to be accessible via its own slave address, a non-zero slave address must be programmed into the `addr` bit field in the `Z2_I2CADDR` register (see Table 4.53) and the `ack` and `enI2C` bits in the `Z2_I2CCTRL` register must be set to 1.

For the ZSSC1956 IBS to be accessible via the global call address (only write access allowed; read access is rejected), the `gc` bit in the `Z2_I2CADDR` register and the `ack` and `enI2C` bits must be set to 1. When the slave module detects a START (or RESTART) condition on the bus, it enables its address checker.

4.10.4.1. Slave Receiver

After the slave detects its own slave address and a write command (see Figure 4.11) and after the slave returns an ACK, an interrupt with the status S_I2cStRxWrAddr is generated (see page 188) and the module becomes the slave receiver. It then expands the low phase of the SCL of the acknowledge bit until its interrupt is cleared. As there is no required content in the data register, software only needs to set the `ack` bit in the `Z2_I2CCTRL` register to the desired value and clear the interrupt (`irq` bit in the `Z2_I2CCTRL` register). The programmed acknowledge bit will be used as the response to the following data byte to be written.

When the `ack` bit is set to 1, an ACK will be returned, indicating that the module is able to receive further data bytes. The master can then send a data byte, which will be acknowledged, and the slave generates a new interrupt after responding with ACK with the status S_I2cStRxWrData (see page 189). Additionally, the low phase of the acknowledge bit on SCL is expanded until the interrupt is cleared. When the interrupt is active, software must read the received data byte first before setting the `ack` bit to the desired value and clearing the interrupt (`irq` bit).

When the ack bit is to set to 0, a NACK will be returned in response to the following data byte, indicating that the module is not able to receive further data and is leaving the bus. After the NACK has been sent, an interrupt with status S_I2cStRxWrDataN is generated (see page 190). Software must read the received data byte first and then must set the ack bit to the desired value and clear the interrupt (irq bit). As the slave leaves the bus, it does not expand the low phase of SCL, but if a new START condition is detected while the interrupt is still active, the low phase of SCL following this START will be expanded so that the slave can setup the ack bit for the next incoming address.

When the master generates a STOP or RESTART condition instead of sending a data byte, an interrupt with status S_I2cStRxSlvEnd is generated immediately (see page 191). If a RESTART or a START follows the STOP, the low phase of SCL following the RESTART/START will be expanded until the interrupt is cleared to be able to setup the ack bit correctly as it might be possible that software has programmed a NACK for the next data byte but wants to return an ACK when it detects its address.

When detecting the global call address and a write command and returning an ACK, the behavior is the same as for being accessed via the slave's own address. Only the status values are different: S_I2cStRxGcAddr instead of S_I2cStRxWrAddr, S_I2cStRxGcData instead of S_I2cStRxWrData, and S_I2cStRxGcDataN instead of S_I2cStRxWrDataN.

A conflict on the I²C™ bus can only be detected during a returned acknowledge bit when the slave is sending a NACK. As the slave leaves the bus after transmitting the NACK, no specific actions are required and the conflict is ignored.

Important: Always clear the interrupt flag to avoid this device blocking the bus.

4.10.4.2. Slave Transmitter

After the slave detects its own slave address and a read command (see Figure 4.10) and after the slave returns an ACK, an interrupt with the status S_I2cStRxRdAddr is generated (see page 191) and the module becomes the slave transmitter. It then expands the low phase of the acknowledge bit on SCL until its interrupt is cleared. Additionally the bus is turned over to the slave for transmitting data. Software must program the data byte to be transmitted into the register Z2_I2CDATA first (see Table 4.56) and then must set the ack bit in the Z2_I2CCTRL register to the desired value and clear the interrupt (irq bit). The programmed acknowledge bit will be used to signal to the hardware whether the programmed data byte is the last one (ack bit == 0) or if further data could be sent (ack bit == 1).

When the received acknowledge bit is an NACK, the master signals that it does not want to read more data and that it wishes to return the bus to the master. An interrupt with status S_I2cStSlvTxDataN is generated (see page 192) and the slave leaves the bus. If a new RESTART/START occurs while the interrupt is still active, the low phase of SCL following the RESTART/START will be expanded until the interrupt is cleared.

When the received acknowledge bit is an ACK but the ack bit is 0, indicating that the sent byte was the last byte, an interrupt with status S_I2cStSlvTxDataL is generated (see page 193) and the slave leaves the bus. The next bytes read by the master will be FF_{HEX} as the slave stops driving the data line. If a new RESTART/START occurs while the interrupt is still active, the low phase of SCL following the RESTART/START will be expanded until the interrupt is cleared.

© 2016 Integrated Device Technology, Inc.

When the received acknowledge bit is an ACK and the ack bit is 1, an interrupt with status S_I2cStSlvTxData is generated (see page 192). The next data byte to be transmitted must be programmed to register Z2_I2CDATA first and then the ack bit must be set to the desired value and the interrupt must be cleared. To avoid the master continuing to generate the clock, this slave extends the low phase of SCL following the ACK until the interrupt is cleared.

A conflict on the bus can only be detected during transmission of a data byte when sending a logic 1 but detecting a logic 0 on the bus, which means that two slaves have the same slave address. In this case, the slave immediately leaves the bus and generates an interrupt with status S_I2cStConflict at the negative clock edge of the acknowledge bit.

4.10.5. Operating as Single Master

When the ZSSC1956 I²C™ module is operating as the only master on the I²C™ bus, all transfers on the bus are started and stopped by this module. Additionally, this module generates the clock for all transfers on the I²C™ bus on the SCL line. The clock can be configured via registers Z2_I2CCLKRATE and Z2_I2CCLKRATE2 (see Table 4.51 and Table 4.52). However, each connected slave is allowed to extend the low phase of the clock, which results in a reduced data rate.

As all transfers are started by this module, no slave address needs to be programmed to the addr bit field in the Z2_I2CADDR register and the global call address does not need to be enabled via the gc bit. This module only needs to be enabled by setting the enI2C bit in the Z2_I2CCTRL register to 1 and setting its multi bit to 0 for a faster bus access time (see section 4.10.8).

To start a transfer as a master, the start bit in the Z2_I2CCTRL register must be set to 1. When the START condition has been generated on the bus and SCL is low, an interrupt is generated with status S_I2cStTxStart (see page 185). When the interrupt is active, the data register must be written with the address of the slave to be accessed and the command bit. Next the stop, start, and irq bits in the Z2_I2CCTRL register must be cleared to continue the transfer. (The start and stop bits must be set to 0 as generating a RESTART or STOP condition on the bus directly after the START is not allowed.) After transmitting the address and the command and after the reception of the acknowledge bit, the next interrupt is generated. The status depends on the transmitted command (read or write) and the received acknowledge bit (ACK or NACK).

4.10.5.1. Master Transmitter

When a write command was sent and a NACK was received, the interrupt status is S_I2cStTxWrAddrN (see page 186). This means that the slave is not ready to receive data. Software then must generate a STOP on the bus by setting the stop bit in the Z2_I2CCTRL register to 1 and clearing the interrupt or must generate a RESTART on the bus by setting the start bit to 1 and clearing the interrupt. When both the stop and start bits are set to 1, first a STOP condition will be generated on the I²C™ bus followed by a START condition. No interrupt will occur for generating a STOP condition. For both RESTART and START, an interrupt with status S_I2cStTxStart occurs.

When a write command has been sent and an ACK has been received, the interrupt status is S_I2cStTxWrAddr (see page 185). This means that the slave is ready to receive data. Software now can send data, or it can generate a STOP or RESTART condition. The behavior for sending a STOP or RESTART is as described above when receiving a NACK. To send data, the byte to be transmitted must be programmed into the data register. Next the `stop`, `start`, and `irq` bits in the `Z2_I2CCTRL` register must be cleared. Depending on the acknowledge bit received as a response to the data byte, an interrupt with status S_I2cStMstTxData (ACK; see page 186) or S_I2cStMstTxDataN (NACK; see page 186) is generated. For a received ACK, the same actions as for the received ACK in response to the address byte can be performed. For a received NACK, the same actions as for the received NACK in response to the address byte can be performed.

4.10.5.2. Master Receiver

When a read command has been sent and a NACK was received, the interrupt status is S_I2cStTxRdAddrN (see page 187). This means that the slave is not ready to deliver data and the master remains the owner of the bus. Software then must generate a STOP on the bus by setting the `stop` bit in the `Z2_I2CCTRL` register to 1 and clearing the interrupt or must generate a RESTART on the bus by setting the `start` bit to 1 and clearing the interrupt. When both the `stop` and `start` bits are set to 1, first a STOP condition will be generated followed by a START condition. No interrupt will occur for generating a STOP condition. For both RESTART and START, an interrupt with status S_I2cStTxStart occurs.

When a read command has been sent and an ACK has been received, the interrupt status is S_I2cStTxRdAddr (see page 187). This means that the slave is ready to deliver data, and the I²C™ bus is turned over to the slave. Software must keep the `start` and `stop` bits low as this module is not the owner of the data line. The `ack` bit in the `Z2_I2CCTRL` register must be set to the appropriate value that will be sent in response to the subsequent received data byte. A value of 0 (NACK will be sent) signals to the slave that the data byte is the last one to be read and the slave must leave the bus so that the master can generate a STOP or a RESTART. A value of 1 (ACK will be sent) signals to the slave that additional bytes will be read afterward. The `irq` bit must also be cleared to continue the transfer.

When a data byte has been received and an ACK has been returned, an interrupt with status S_I2cStMstRxData occurs (see page 187) and the slave retains the ownership of the data line. Software must first read the received byte and then must set the acknowledge bit to the desired value and to clear the interrupt bit.

When a data byte has been received and an NACK has been returned, an interrupt with status S_I2cStMstRxDataN occurs (see page 188). The bus is returned to the master. Software must read the received data byte and then must generate a STOP or RESTART by setting the appropriate bits and must clear the interrupt.

4.10.6. Operating as Master on a Multi-Master Bus

When the ZSSC1956 I²C™ module is operating as one of many masters on the I²C™ bus, the transfers on the I²C™ bus can be started by this or by another module. When this module is the master, it generates the clock for its master transfers on the SCL line using registers `Z2_I2CCLKRATE` and `Z2_I2CCLKRATE2` while it will use the clock provided by another master if accessed as slave. When two masters drive the clock line at the same time before any contention has occurred, the high phase is determined by the fastest master on the bus while the low phase is determined by the slower one. However, each connected slave is allowed to extend the low phase of the clock, which results in a reduced data rate.

To be accessible via its own slave address, a non-zero slave address must be programmed into the addr bit field in the Z2_I2CADDR register and the ack bit and enI2C bits in the Z2_I2CCTRL register must be set to 1. To be able to be accessed via the global call address (only write access allowed, read access is rejected), the gc bit in the Z2_I2CADDR register and the ack and enI2C bits must be set to 1. When the slave module detects a START (or RESTART) condition on the bus, it enables its address checker. When this module is enabled by writing 1 to the enI2C bit, then the multi bit must also be set to 1 as there might already be an active transfer by another master. This is needed to avoid this module disturbing the traffic on the bus (see section 4.10.8).

The handling for transfers is almost the same as described in the previous sections. Only some additional status interrupts can occur due to conflict situations. On a multi-master bus, it is possible that at least two masters assume the bus is free and generate a START condition at the same moment.

If the master detects a conflict during the address and command byte, it immediately switches into slave mode to check whether it will be accessed or not. In this situation, if it detects its own slave address and a read command and then it returns an ACK (ack bit == 1), an interrupt with status S_I2cStRxRdAddrL (see page 192) instead of status S_I2cStRxRdAddr is generated and the device becomes a slave transmitter. In this situation, if it detects its own slave address and a write command and then it returns an ACK (ack bit == 1), an interrupt with status S_I2cStRxWrAddrL (see page 188) instead of status S_I2cStRxWrAddr is generated and the device becomes a slave receiver. In this situation, if it detects the global call address and a write command and then it returns an ACK (ack bit == 1), an interrupt with status S_I2cStRxGcAddrL (see page 189) instead of status S_I2cStRxGcAddr (see page 189) is generated and the device becomes a slave receiver. Otherwise (wrong address, NACK returned), an interrupt with status S_I2cStConflict is generated and the device leaves the bus.

It is also possible that two masters could access the same slave with the same command. In this case, a conflict might be detected during the transmission of a data byte or an acknowledge bit. In both cases, an interrupt with status S_I2cStConflict is generated and the device leaves the bus.

4.10.7. Error Conditions

In addition to all the interrupts related to transmission and reception, two error interrupts are possible. If one of the state machines in this module enters an undefined state (due to cosmic radiation), an interrupt with status S_I2cStHWError (see page 194) is generated. To solve this situation, the module must be disabled as all state machines are then set back to their default states.

If a START or STOP condition is detected on the bus during an active transfer as a master or a slave, this module immediately leaves the bus and generates an interrupt with status S_I2cStBusError (see page 194). If this error was caused by a START condition or if a START conditions follows afterward, the low phase of SCL following the START will be expanded until the interrupt is cleared to be able to setup the device correctly for a new transfer.

Warning: Due to several error conditions, for example if a slave missed one clock cycle, other state transitions are possible, and the bus might get stuck. When an unexpected state transition occurs, the module must be disabled and re-enabled to clean up the bus. One possible situation is a conflict when operating as single master.

4.10.8. Bus States

There are four different states possible for the I^{2}C™ bus. The bus can be busy, free, or stuck, or the state can be unknown. The last state occurs when the device is just enabled. The determination of the bus state depends on whether the device is the only master on the bus (multi bit in the Z2_I2CCTRL register == 0) or not (multi bit == 1).

4.10.8.1. Single Master

At enable, the I²C™ bus state is unknown as the module did not observe the bus when it was disabled. When this device is configured to be the only master on the bus (`multi` bit in the `Z2_I2CCTRL` register == 0), no START or STOP condition can occur. Additionally SCL cannot change from high to low as this transition is only allowed for masters, but a slave can only hold SCL low after it has detected a low clock line. This is needed to handle interrupts and data before the transfer continues. Therefore three bus conditions are possible after enabling this module:

- SCL == 1 and SDA == 1: When both lines are undriven, the module assumes the bus to be free.
- SCL == 1 and SDA == 0: This situation can only occur if a slave has missed a clock pulse or if the master was disabled within a transfer. In this case, the module assumes the bus to be stuck and generates clock pulses until both lines are undriven.
- SCL == 0: This situation can only occur when the master was disabled within a transfer and the accessed slave holds SCL low as it has not cleared its interrupt. This situation cannot be directly cleared so the master must wait until the slave releases SCL. After that, one of the two situations above is present which the master is able to handle.

Note: The module assumes the bus is free if both SCL and SDA are high. When the module detects a START condition (generated by itself), it assumes the bus is busy.

Note: If the bus is stuck, the device generates clock pulses until both SDA and SCL are high. Then it assumes the bus is free again. Therefore the `multi` bit must be set to 1 when operating as slave-only to avoid this module assuming the bus to be stuck and generating clock pulses on the SCL line.

Note: The bus is only busy after the module has generated a START. Normally, the module generates a STOP to release the bus at the end of its transfer. When this STOP does not occur on the bus because a slave drives SDA low due to an error, the bus is stuck.

4.10.8.2. Multi Master

At enable, the bus state is unknown as the module did not observe the bus when it was disabled. In contrast to a single master system, it cannot assume the bus to be free when both lines are undriven as this situation also occurs during the transmission of a 1 when the clock is high. The following conditions are possible:

- START detected: The module assumes the bus to be busy.
- STOP detected: The bus assumes the bus to be free.
- Falling edge of SCL: As only a master is allowed to change the level of SCL from 1 to 0, the module assumes an active transfer and therefore the bus to be busy.
- SCL == 1 and SDA == 1: When this condition is true for a specific time (timeout), the module assumes the bus to be free. Before the timeout occurs, the bus state remains unknown.
- SCL == 1 and SDA == 0: When this condition is true for a specific time (timeout), the module assumes the bus to be stuck. Before the timeout occurs, the bus state remains unknown.
- SCL == 0: This situation can occur because a transfer is active but also can occur due to the situation described for single master. Therefore the master cannot determine the correct state and waits until SCL becomes high again.

Note: The module assumes the bus is free if both SCL and SDA are high. When the module detects a START condition, it assumes the bus is busy.

Note: If the bus is stuck, the device generates clock pulses until both SDA and SCL are high. Then it assumes the bus is free again.

Note: The bus is busy after a detection of a START condition generated by any master. The following conditions are possible for a change of the bus state:

- STOP detected: The bus assumes the bus to be free.
- SCL == 1 and SDA == 1: When this condition is true for a specific time (timeout), the module assumes the bus to be free.
- SCL == 1 and SDA == 0: When this condition is true for a specific time (timeout), the module assumes the bus to be stuck.
- STOP sent but not detected on the bus: The module assumes the bus to be stuck.

4.10.9. Status Description

Status name and code: S_I2cStIdle – 00_{HEX}

- **Description:** This is the only state where no interrupt is activated. This state is entered via reset if the device is disabled or when this device is the master and a STOP condition is generated.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: 0
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Note: This state is entered without activating the interrupt line.

Status name and code: S_I2cStTxStart – 01_{HEX} (MASTER STATE)

- **Description:** This state is entered after a START condition was generated on the bus. The interrupt is entered after SCL is low. SCL stays low until the interrupt is cleared. The device has allocated the bus as master and will continue generating the clock.
- **First action (access to `Z2_I2CDATA` register):** Program command (R/W; bit 0) and address (bits [7:1]) of slave to be accessed.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: Clear
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxWrAddr
  - S_I2cStTxWrAddrN
  - S_I2cStTxRdAddr
  - S_I2cStTxRdAddrN
  - S_I2cStRxRdAddrL
  - S_I2cStRxGcAddrL
  - S_I2cStRxWrAddrL
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Note: Generating a STOP or RESTART condition directly after a START condition is not allowed.

Status name and code: S_I2cStTxWrAddr – 02_{HEX} (TX MASTER STATE)

- **Description:** This state is entered after ACK is received as a response to a successful transmission of the slave address and write command. The ACK response means that the slave is ready to receive data.
- **First action (access to `Z2_I2CDATA` register):** Program data to be written to the slave only if data will be transmitted but not when RESTART or STOP will be generated.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStMstTxData
  - S_I2cStMstTxDataN
  - S_I2cStTxStart
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable / STOP)

Status name and code: S_I2cStTxWrAddrN – 03_{HEX} (TX MASTER STATE)

- **Description:** This state is entered after NACK is received as a response to a successful transmission of the slave address and write command. The NACK response means that the slave is not ready to receive data or the address used is not assigned to any slave on the bus.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStIdle (disable / STOP)

Status name and code: S_I2cStMstTxData – 04_{HEX} (TX MASTER STATE)

- **Description:** This state is entered after ACK is received as the response to a successful transmission of data. The ACK response means that the slave is ready to receive more data.
- **First action (access to `Z2_I2CDATA` register):** Program data to be written to the slave only if data will be transmitted but not when RESTART or STOP will be generated.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStMstTxData
  - S_I2cStMstTxDataN
  - S_I2cStTxStart
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable / STOP)

Status name and code: S_I2cStMstTxDataN – 05_{HEX} (TX MASTER STATE)

- **Description:** This state is entered after NACK is received as a response to a successful transmission of data. The NACK response means that the slave is not able to receive more data.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStIdle (disable / STOP)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStTxRdAddr – 06_{HEX} (RX MASTER STATE)

- **Description:** This state is entered after ACK is received as a response to a successful transmission of the slave address and read command. The ACK response means that the slave is ready to transmit data → bus turnaround, slave becomes the transmitter.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStMstRxData
  - S_I2cStMstRxDataN
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStTxRdAddrN – 07_{HEX} (RX MASTER STATE)

- **Description:** This state is entered after NACK is received as a response to a successful transmission of the slave address and read command. The NACK response means that the slave is not ready to transmit data or the address used is not assigned to any slave on the bus → no bus turnaround, the master keeps the bus to generate the STOP or RESTART.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStIdle (disable / STOP)

Status name and code: S_I2cStMstRxData – 08_{HEX} (RX MASTER STATE)

- **Description:** This state is entered after ACK is transmitted as a response to a received data byte. The ACK response means that the slave remains the transmitter and that the master waits for the next data byte.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStMstRxData
  - S_I2cStMstRxDataN
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStMstRxDataN – 09_{HEX} (RX MASTER STATE)

- **Description:** This state is entered after NACK is transmitted as a response to a received data byte. The NACK response means that the slave releases the bus (bus turnaround) and that the master becomes the transmitter for STOP or RESTART generation.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0 or 1
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStIdle (disable / STOP)

Status name and code: S_I2cStRxWrAddr – $0A_{HEX}$ (RX SLAVE STATE)

- **Description:** This state is entered when the device's own slave address and a write command were received and ACK was returned while the device was idle.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxWrData
  - S_I2cStRxWrDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxWrAddrL – $0B_{HEX}$ (RX SLAVE STATE; multi-master only)

- **Description:** This state is entered when the device's own slave address and a write command were received and ACK was returned while the device lost arbitration. This can happen after an S_I2cStTxStart interrupt when another master also accessed the bus, winning arbitration and accessing this device.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxWrData
  - S_I2cStRxWrDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStRxGcAddr – $0C_{HEX}$ (RX SLAVE STATE)

- **Description:** This state is entered when the global call address and a write command were received and ACK was returned while the device was idle.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxWrData
  - S_I2cStRxWrDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxGcAddrL – $0D_{HEX}$ (RX SLAVE STATE; multi-master only)

- **Description:** This state is entered when the global call address and a write command were received and ACK was returned while the device lost arbitration. This can happen after an S_I2cStTxStart interrupt when another master also accessed the bus, winning arbitration and accessing this device.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxWrData
  - S_I2cStRxWrDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxWrData – $0E_{HEX}$ (RX SLAVE STATE)

- **Description:** This state is entered when the device was accessed via its slave address, data was received, and ACK was returned.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxWrData
  - S_I2cStRxWrDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStRxWrDataN – $0F_{HEX}$ (RX SLAVE STATE)

- **Description:** This state is entered when the device is accessed via its slave address, data was received, and NACK was returned. The device leaves the active slave state.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxGcData – 10_{HEX} (RX SLAVE STATE)

- **Description:** This state is entered when the device was accessed via the global call address, data was received, and ACK was returned.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStRxSlvEnd
  - S_I2cStRxGcData
  - S_I2cStRxGcDataN
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxGcDataN – 11_{HEX} (RX SLAVE STATE)

- **Description:** This state is entered when the device is accessed via the global call address, data was received, and NACK returned. The device leaves the active slave state.
- **First action (access to `Z2_I2CDATA` register):** Read received data byte.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStRxSlvEnd – 12_{HEX} (RX SLAVE STATE)

- **Description:** This state is entered when the device is operating as an active slave and a STOP or RESTART condition is received.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStRxRdAddr – 13_{HEX} (TX SLAVE STATE)

- **Description:** This state is entered when the device's own slave address and a read command were received and ACK was returned while the device was idle.
- **First action (access to `Z2_I2CDATA` register):** Program data to be written to master.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStSlvTxData
  - S_I2cStSlvTxDataN
  - S_I2cStSlvTxDataL
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

Status name and code: S_I2cStRxRdAddrL – 14_{HEX} (TX SLAVE STATE; multi-master only)

- **Description:** This state is entered when the device's own slave address and a read command were received and ACK was returned while the device lost arbitration. This can happen after an S_I2cStTxStart interrupt when another master also accessed the bus, winning arbitration and accessing this device.
- **First action (access to `Z2_I2CDATA` register):** Program data to be written to master.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStSlvTxData
  - S_I2cStSlvTxDataN
  - S_I2cStSlvTxDataL
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStSlvTxData – 15_{HEX} (TX SLAVE STATE)

- **Description:** This state is entered when data was transmitted, ACK was returned by the master, and there is still data to be transmitted (`ack` bit in the `Z2_I2CCTRL` register == 1).
- **First action (access to `Z2_I2CDATA` register):** Program data to be written to master.
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStSlvTxData
  - S_I2cStSlvTxDataN
  - S_I2cStSlvTxDataL
  - S_I2cStConflict
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStSlvTxDataN – 16_{HEX} (TX SLAVE STATE)

- **Description:** This state is entered when data was transmitted and NACK was returned by the master. The device leaves the active slave state.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1

- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStSlvTxDataL – 17_{HEX} (TX SLAVE STATE)

- **Description:** This state is entered when data was transmitted, ACK was returned by the master, but no more data is available (`ack` bit in the `Z2_I2CCTRL` register == 0). The device leaves the active slave state.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStConflict – 18_{HEX} (MASTER / SLAVE STATE)

- **Description:** This state is entered when the device is sending a logic 1 but receiving a logic 0. This can happen when
  - sending the slave address or the read command as the master device
  - sending data as the master transmitter
  - sending NACK as the master receiver
  - sending data as the slave transmitter
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Note: When sending NACK as the slave receiver, it is not interpreted as a conflict. The bus is just left. (The master might write to multiple slaves.)

Status name and code: S_I2cStBusError – 19_{HEX} (MASTER / SLAVE STATE)

- **Description:** This state is entered when device is active as the master or slave and a RESTART/START or STOP condition is detected at a wrong position within a transfer. When detecting an error, the device leaves the bus immediately.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0 or 1
  - Irq:: Clear
  - Ack: 0 or 1
- **Possible next status:**
  - S_I2cStTxStart
  - S_I2cStRxRdAddr
  - S_I2cStRxGcAddr
  - S_I2cStRxWrAddr
  - S_I2cStBusError
  - S_I2cStIdle (disable)

Status name and code: S_I2cStHWError – $1F_{HEX}$ (MASTER / SLAVE STATE)

- **Description:** This state is entered when one of the state machines goes to an undefined state (due to cosmic radiation). This state is used for safety only. To leave this state and to setup the module correctly, the module must be disabled and re-enabled again.
- **First action (access to `Z2_I2CDATA` register):** ---
- **Second action (write to `Z2_I2CCTRL` register):**
  - Stop: 0
  - Start: 0
  - Irq:: Clear
  - Ack: 0
- **Possible next status:**
  - S_I2cStIdle (disable)

© 2016 Integrated Device Technology, Inc.

4.10.10. egister Overview for I²C™ Module

4.10.10.1. Register "Z2_I2CCLKRATE" and "Z2_ I2CCLKRATE 2 – Baud Rate Configuration

Table 4.51 Register Z2_I2CCLKRATE – system address 4000 1C20$_{HEX}$

Name	Bits	Default	Access	Description
crLsb	[7:0]	FF_{HEX}	RW	Configuration of bit rate for master operation. bit rate = clk / (2*[{crMsb, crLsb}+1]) **Note:** Do not program values less than 7. **Note:** Do not change during active master transfer.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

Table 4.52 Register Z2_I2CCLKRATE2 – system address 4000 1C24$_{HEX}$

Name	Bits	Default	Access	Description
crMsb	[5:0]	$11\ 1111_{BIN}$	RW	Configuration of bit rate for master operation. Bit rate = clk / (2*[{crMsb, crLsb}+1]) **Note:** Do not program values less than 7. **Note:** Do not change during active master transfer.
Unused	[31:6]	$0…0_{BIN}$	RO	Unused; always write as 0.

4.10.10.2. Register "Z2_I2CADDR" – I²C™ Address

Table 4.53 Register Z2_I2CADDR – system address 4000 1C28$_{HEX}$

Name	Bits	Default	Access	Description
gc	[0]	0_{BIN}	RW	General call address enable. A write access by another master using the general call address is only accepted when this bit is set to 1. **Note:** Read accesses using the general call address are not allowed and ignored by hardware.
addr	[7:1]	0_{BIN}	RW	Own I^2C™ slave address. Used to recognize if another master tries to access this device.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

4.10.10.3. Register "Z2_I2CCTRL" – I²C™ Control

Table 4.54 Register Z2_I2CCTRL – system address 4000 1C2C$_{HEX}$

Name	Bits	Default	Access	Description
enI2C	[0]	0_{BIN}	RW	Enable bit for the I²C™ module; all state machines are reset at disable. **Note:** Do not disable the I²C™ module during an active master transfer. Otherwise a slave could block the bus if it did not receive enough clock pulses.
multi	[1]	0_{BIN}	RW	This bit must be set in multi-master applications. When set to 1, the timeout counter to detect a stuck bus and to detect a free or busy bus state at enable of the I²C™ module is enabled. **Note:** Change only when module is disabled.
ack	[2]	0_{BIN}	RW	When set to 1, an ACK bit is generated in response to a detected address match (slave access) as well as a response to a received data byte (master and slave). It is also used to stop the transfer as a slave transmitter (see status handling). **Note:** When the ack bit is not set, no packet will be received; when set, packets with matching addresses or global addresses will be received if enabled.
irq	[3]	0_{BIN}	RW	Interrupt bit. This bit is set by hardware and must be cleared by software after handling the interrupt. All transactions must be interrupt-driven.
stop	[4]	0_{BIN}	RW	Stop bit for master mode. This bit must be set by software to stop a master transfer and to generate a STOP on the bus. It is cleared by hardware.
start	[5]	0_{BIN}	RW	Start or restart bit for master mode. This bit must be set by software to generate a START or RESTART condition on the bus. This bit must be cleared by software after each interrupt.
Unused	[31:6]	$0…0_{BIN}$	RO	Unused; always write as 0.

4.10.10.4. Register "Z2_I2CSTAT" – I²C™ Status

Table 4.55 Register Z2_I2CSTAT – system address 4000 1C30$_{HEX}$

Name	Bits	Default	Access	Description
gc_stat	[4:0]	00000_{BIN}	RO	This value reflects the last interrupt reason (see section 4.10.9): 00_{HEX}: S_I2cStIdle 01_{HEX}: S_I2cStTxStart 02_{HEX}: S_I2cStTxWrAddr 03_{HEX}: S_I2cStTxWrAddrN 04_{HEX}: S_I2cStMstTxData 05_{HEX}: S_I2cStMstTxDataN 06_{HEX}: S_I2cStTxRdAddr 07_{HEX}: S_I2cStTxRdAddrN 08_{HEX}: S_I2cStMstRxData 09_{HEX}: S_I2cStMstRxDataN $0A_{HEX}$: S_I2cStRxWrAddr $0B_{HEX}$: S_I2cStRxWrAddrL $0C_{HEX}$: S_I2cStRxGcAddr $0D_{HEX}$: S_I2cStRxGcAddrL $0E_{HEX}$: S_I2cStRxWrData $0F_{HEX}$: S_I2cStRxWrDataN 10_{HEX}: S_I2cStRxGcData 11_{HEX}: S_I2cStRxGcDataN 12_{HEX}: S_I2cStRxSlvEnd 13_{HEX}: S_I2cStRxRdAddr 14_{HEX}: S_I2cStRxRdAddrL 15_{HEX}: S_I2cStSlvTxData 16_{HEX}: S_I2cStSlvTxDataN 17_{HEX}: S_I2cStSlvTxDataL 18_{HEX}: S_I2cStConflict 19_{HEX}: S_I2cStBusError $1F_{HEX}$: S_I2cStHWError
Unused	[31:5]	$0…0_{BIN}$	RO	Unused; always write as 0.

4.10.10.5. Register "Z2_I2CDATA" – I²C™ Data

Table 4.56 Register Z2_I2CDATA – system address 4000 1C34$_{HEX}$

Name	Bits	Default	Access	Description
gc_data	[7:0]	00_{HEX}	RW	Data register. As there are no buffers for RX or TX, data is shifted through this register. **Note:** Write access is only allowed (and possible) when the I²C™ module is enabled and the interrupt is active; additional restrictions occur due to the interrupt reason (status). **Note:** Read access is only valid if the interrupt is active.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

4.11. USART in ZSYSTEM2

The USART module provides four different operating modes. Two of them deliver synchronous operation with a size of 8 bits, during which this module is the master (it generates the clock). These modes vary only in the sampling position in the RX mode. The other two modes offer asynchronous operation: one with a size of 8 bits and the other with a size of 9 bits. The last mode can also be used for multiprocessor communication.

The maximum possible baud rate on the bus is a fifth of the internal clock in synchronous mode or a fourth of the internal clock in asynchronous mode.

The USART has an active-high interrupt line connected to ARM® interrupt 7.

Important: Before the USART can be used, the clock of ZSYSTEM2 must be enabled first via register SYS_CLKCFG (see Table 4.5). After the clock for the ZSYSTEM2 is enabled, the USART pins must be mapped onto appropriate GPIO pads (see section 4.3.4).

The USART module in ZSYSTEM2 has an active-high interrupt line connected to ARM® interrupt 7.

4.11.1. External Signal Lines

The USART bus consists of two external signal lines, TXD and RXD. The TXD line always operates as an output. Therefore the behavior of its output driver (open-drain; push-pull) is selectable by setting the corresponding bit of bit field ppNod of register SYS_MEMPORTCFG (see Table 4.6) to the desired value. In synchronous mode, the TXD line is used to provide the clock for the connected slave. In asynchronous mode, the TXD line is used to send data to the connected device. As the RXD line is used to send and receive data in synchronous mode, its output driver always operates as an open-drain independent of the setting of the ppNod bit field. In asynchronous mode, it is always used to receive data from the connected device.

4.11.2. Asynchronous Mode

When operating in asynchronous mode (Mode bit field in register Z2_USARTCFG is set to 2 or 3; see Table 4.57), the receive channel and the transmit channel are completely independent. When mode 2 is selected, 8 bits are transferred between the START bit and the STOP bit. In mode 3, there are 9 bits transferred between the START bit and the STOP bit. The USART module is enabled by the UsartEn bit in register Z2_USARTCFG. The mode must not be changed when the module is enabled, but it can be selected in the same write access as the module enable.

As the operation is asynchronous, the timing of the transfer must be defined identically on both sides (this device and the connected device) before a transfer can take part. This can be done for this device by programming the appropriate value to the baud rate registers (Z2_USARTCLK1 and Z2_USARTCLK2; see section 4.11.4.5). The baud rate depends on the divided system clock (see section 4.3.2) and must only be changed when the module is disabled. The value to be programmed is calculated using the following formula:

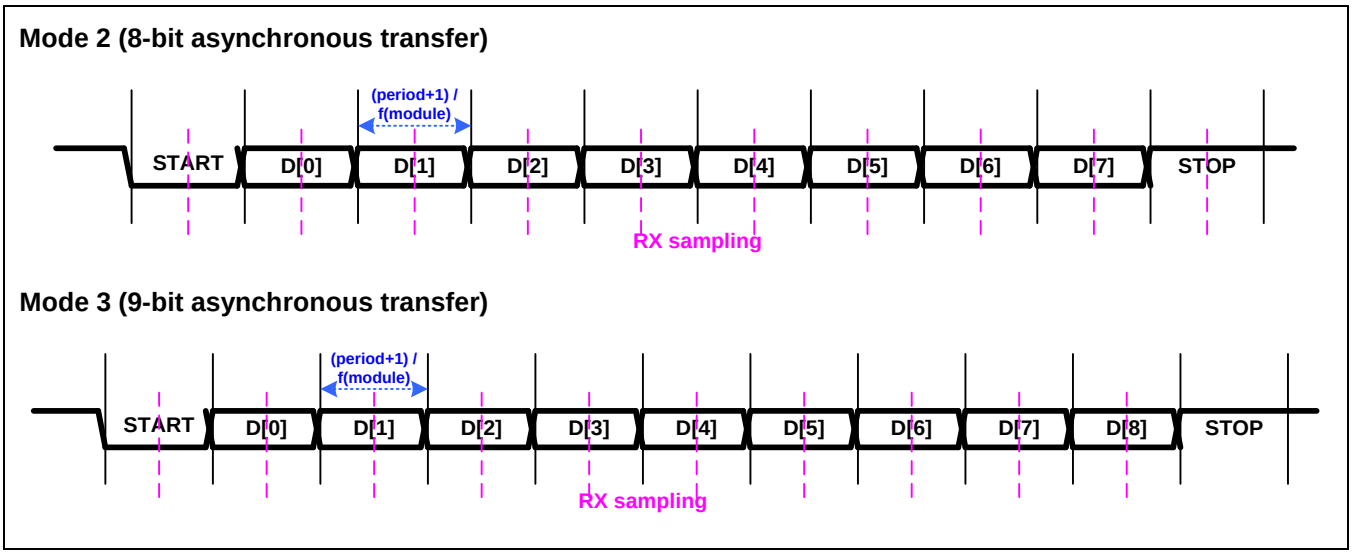
$$\text{period} = \text{round}(\text{module clock} / \text{baud rate}) - 1$$

The minimum value allowed to be programmed is 3.

While there is no dedicated enable for the TX channel (transmission is started by a write access to the TX buffer), the receive channel has a dedicated enable bit (RxEn_USART bit in register Z2_USARTCFG). As the incoming data cannot be influenced regarding arrival time, it is recommended that enabling or disabling the receive channel only be performed when the module is disabled. The RX enable bit can be selected in the same write access as the module enable.

When the module is disabled, all state machines as well as all status bits in the Z2_USARTSTAT register except RxFull_USART and RxBit8 are set back to their default state (see Table 4.58). The RxFull_USART status bit is only cleared by read access to the RX buffer to prevent losing data at disable.

Figure 4.12 Data Format of Asynchronous Transfers



4.11.2.1. USART Asynchronous Transmission

A transmission is started by writing the data to be transmitted into the TX buffer (write to Z2_USARTDATA; see Table 4.59). As this register is only 8-bit wide, the ninth bit (MSB) in mode 3 must be written to TxBit8 (Z2_USARTCFG [7]) before writing the LSBs to the TX buffer. Writing to the TX buffer clears the status flag TxBufEmpty in the status register Z2_USARTSTAT (see Table 4.58). When the transmitter is idle (bit TxSrEmpty is 1), the START bit and the STOP bit are added to the TX data and all 10/11 bits are moved into the TX shift register. The status flag TxBufEmpty is cleared, and the data is shifted out of the module. Both the START bit and the STOP bit have a length of 1 bit. New data can be written into TxBit8 and into the TX buffer when the buffer is marked as empty (directly after the data transfer into the TX shift register).

When the data is shifted out and further data is present in the TX buffer, the next transfer follows immediately. When no further data is present, the transmitter stops and the TxSrEmpty flag is set. When the software tries to write to the TX buffer or to change TxBit8 while the buffer is not empty (bit TxBufEmpty is 0), the write access is rejected (old data is kept) and the write collision flag WrCollision_USART is set to 1.

All three flags (TxSrEmpty, TxEmpty_USART, WrCollision_USART) are allowed to drive the interrupt line when they are enabled for this via register Z2_USARTIRQEN (see Table 4.60). All three flags are set to their default values when the module is disabled. The flag TxSrEmpty is also set and cleared by hardware only. The flag TxEmpty_USART is set by data transfer into the shift register and cleared by write access to the TX buffer. WrCollision_USART is only set by hardware and cleared on read access to the status register Z2_USARTSTAT.

4.11.2.2. USART Asynchronous Reception

The module synchronizes to incoming data on the falling edge on the RXD line (START detection). After half a period, it is checked whether the value on the RXD line is still 0. If this is not the case, the actual transfer is stopped, the status flag `StartErr` is set to 1 in register Z2_USARTSTAT (see Table 4.58) and the module waits on the next falling edge on the RXD line. This error condition can occur due to a mismatch in the programmed period in both devices, due to a spike on the RXD line that caused the erroneous synchronization or due to a misaligned enable of the module. The last situation could be avoided by software if both devices send FF_{HEX} as data for an initial synchronization. In this case, only the START bit would drive the data line low. The flag `StartErr` is set by hardware and is cleared by read access to the status register Z2_USARTSTAT or when the module is disabled. Further data reception is not blocked when this bit is set. When enabled via Z2_USARTIRQEN (see Table 4.60), this flag is allowed to drive the interrupt line.

When operating in mode 2 and the `Mpce` bit (Z2_USARTCFG[4]) is 0, the received data byte is stored into the RX buffer and the level of the received STOP bit is stored into `RxBit8` (Z2_USARTSTAT[7]). The data is stored at the sampling position of the STOP bit. If the `Mpce` bit is 1 instead, the received byte and STOP bit are only stored when the STOP bit has a value of 1. Otherwise the data is rejected. The rejection is not signaled to the software.

When data is stored in the RX buffer, the `RxFull_USART` flag is set. This flag is cleared by reading the data out of the RX buffer. If the RX buffer is marked as full when new data is received, the new data is rejected and the RxOverflow_USART status flag is set. As there is no overflow check for `RxBit8`, the status including `RxBit8` must be read before the RX buffer.

The flag RxOverflow_USART is set by hardware and cleared by read access to the status register or when the module is disabled. When enabled via register Z2_USARTIRQEN, this flag is allowed to drive the interrupt line. Also the `RxFull_USART` flag is set by hardware and is allowed to drive the interrupt line. This flag is cleared on read access to the RX buffer, but it is not cleared when module is disabled to avoid loss of data.

When operating in mode 3, there will be 9 instead of 8 data bits received. The module behaves almost the same, except that there is no check for the STOP bit level. Instead the level of the ninth data bit can be checked when bit `Mpce` is set to 1. This can be used for multiprocessor communication.

In multiprocessor communication, all devices set their `Mpce` bit to 1. In this case, they only receive and store data when the ninth data bit is 1. The ninth bit is used to distinguish between the address byte (ninth bit is 1) and data bytes (ninth bit is 0). For a complete transfer, the first byte is sent with the ninth bit set to 1 where this byte is used as the address byte. All devices will receive this byte and can do an address check in software. After the address check is performed, the addressed device clears its `Mpce` bit to be able to receive the following data bytes and the originator of the address sends data bytes with the ninth bit set to 0. All other devices that are not addressed, keep their `Mpce` bit set to 1 to ignore all incoming data bytes. They continue receiving when the next address byte arrives.

4.11.3. Synchronous Mode

When operating in synchronous mode (`Mode` bit field in register Z2_USARTCFG is set to 0 or 1; see Table 4.57), this module generates the clock on the TXD line whenever a transfer will take place. Each transfer consists of 8 bits without any START or STOP bit. The data is transferred via the RXD line. The USART module is enabled by the `UsartEn` bit in register Z2_USARTCFG. The mode must not be changed when the module is enabled, but it can be selected in the same write access as the module enable. The two modes differ only by the sampling position.

The transfer speed must be programmed as in asynchronous mode with the following formula:

period = round(module clock / baud rate) – 1

The minimum value allowed to be programmed is 4.

The clock is generated by hardware. The falling edge is generated at approximately ¼ of the period and the rising edge is generated at approximately ¾ of the period.

If the real period (programmed period + 1) is equal to

- 4∗N the high phase of the generated clock is equal to the low phase.
- 4∗N+1 the high phase of the generated clock is one cycle longer than the low phase.
- 4∗N+2 the high phase of the generated clock is equal to the low phase.
- 4∗N+3 the high phase of the generated clock is one cycle shorter than the low phase.

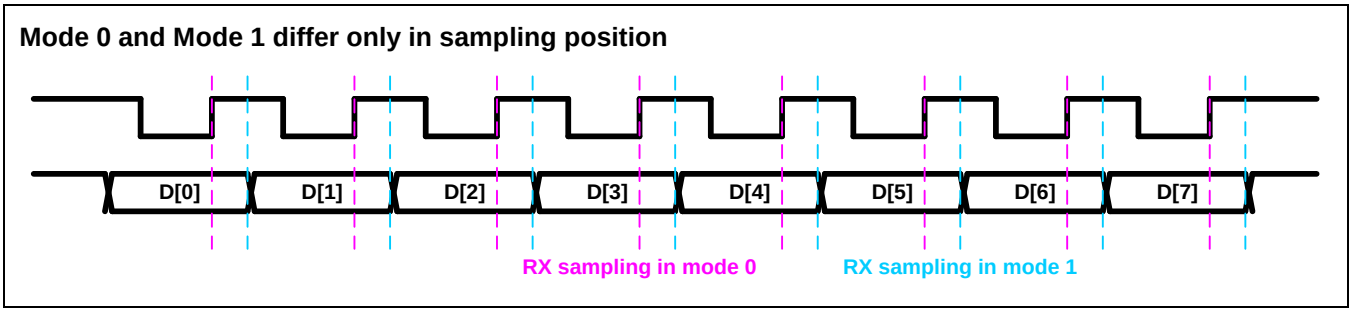
As one line is used for the transfer clock, the direction of the transfer must be selected via the RX enable bit (RxEn_USART in register Z2_USARTCFG). As this module controls the transfer, the RX enable bit should only be changed when no transfer is in progress. This can be checked via the TxSrEmpty status flag.

For transmission, the data byte to be transmitted must be programmed into the TX buffer. The data byte is then transferred into the TX shift register and shifted out of the module. The three status flags behave as in asynchronous mode.

Because the RXD line is a bidirectional open-drain buffer, a receive transfer is started by writing FF_{HEX} into the TX buffer. This means that receiving is the same as transmitting FF_{HEX} except that the RX enable bit (RxEn_USART) is set. The status flags RxOverflow_USART and RxFull_USART behave as in asynchronous mode except that they are set at the sampling position of the last bit of the received data byte.

In mode 0, the RXD line is sampled at the rising edge of the generated clock. As the transfer is synchronous, the connected slave is assumed to send the data to the RX line on the falling edge. Therefore the device samples the data half a period after it was generated. This time is further shortened due to the PCB delay of the clock, the internal delay of the accessed device, and the PCB delay of the returned data. To address this timing issue, a second mode (mode 1) is implemented where the data is sampled later at the end of the bit period.

Figure 4.13 Data Format of Synchronous Transfers



4.11.4. Register Overview of USART

4.11.4.1. Register "Z2_USARTCFG" – USART Configuration

Table 4.57 Register Z2_USARTCFG – system address 4000 1C40$_{HEX}$

Name	Bits	Default	Access	Description
UsartEn	[0]	0_{BIN}	RW	Enable for the USART.
RxEn_USART	[1]	0_{BIN}	RW	Enable for the RX part of the USART. **Note:** RxEn_USART selects the direction in mode 0 and 1; change only when no transfer is in progress. **Note:** Change only when module is disabled in mode 2 or 3.
Mode	[3:2]	00_{BIN}	RW	USART mode. 0: 8 bit synchronous operation; RX sampling at rising edge 1: 8 bit synchronous operation; RX sampling at end of bit period 2: 8 bit asynchronous operation 3: 9 bit asynchronous operation **Note:** Change only when module is disabled.
Mpce	[4]	0_{BIN}	RW	Multiprocessor communication enable; unused in modes 0 and 1. In mode 2: 0: ignore level of STOP bit 1: receive only when STOP bit is 1 In mode 3: 0: ignore level of ninth bit 1: receive only when ninth bit is 1
Unused	[6:5]	00_{BIN}	RO	Unused; always write as 0.
TxBit8	[7]	0_{BIN}	RW	Ninth bit to be transmitted in mode 3; unused in other modes. **Note:** If a write access to this register occurs when the TX buffer is full and when operating in mode 3, this bit keeps its contents and the written bit is rejected. This is signaled by the WrCollision_USART flag in the status register only if write access would have changed this bit.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

4.11.4.2. Register "Z2_USARTSTAT" – USART Status

Table 4.58 Register Z2_USARTSTAT – system address 4000 1C44$_{HEX}$

Name	Bits	Default	Access	Description
RxOverflow_USART	[0]	0_{BIN}	RC	This bit signals that an Rx overflow occurred. **Note:** This bit is cleared when the status is read or when disabling the module. **Note:** The received byte causing the overflow is rejected; the previous received bit is kept in the RX buffer.
RxFull_USART	[1]	0_{BIN}	RO	This bit reflects the status of the RX buffer. It is set when a new byte is transferred into the RX buffer. **Note:** This bit is cleared when `UsartData` is read.
TxBufEmpty	[2]	1_{BIN}	RO	This bit reflects the status of the TX buffer. It is set when a byte is transferred from the TX buffer into the shift register. It is cleared when writing data to TX buffer. **Note:** This bit is set when disabling the module.
WrCollision_USART	[3]	0_{BIN}	RC	This bit is set when `UsartData` is written while TX buffer is already full. **Note:** This bit is cleared when the status is read or when disabling the module.
StartErr	[4]	0_{BIN}	RC	This bit is set when an invalid START bit is detected in mode 2 or 3; the currently active RX transfer is stopped. **Note:** This bit is cleared when the status is read or when disabling the module.
TxSrEmpty	[5]	1_{BIN}	RO	This bit reflects the status of the TX shift register. It is cleared when a byte is transferred from the TX buffer into the shift register. It is set when data is transferred and no more data is available in the TX buffer. **Note:** This bit is set when disabling the module.
RxActive_USART	[6]	0_{BIN}	RO	This bit reflects the status of the receiver in mode 2 and 3. **Note:** This bit is cleared when disabling the module.
RxBit8	[7]	0_{BIN}	RO	This bit reflects the value of the ninth received bit in mode 3 and the value of the STOP bit in mode 2.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

4.11.4.3. Register "Z2_USARTDATA" – USART Data Buffers

Table 4.59 Register Z2_USARTDATA – system address 4000 1C48$_{HEX}$

Name	Bits	Default	Access	Description
UsartData	[7:0]	FF_{HEX}	RW	When writing a byte to this register, the value is stored in the TX buffer. Additionally a write access to this register clears the `TxEmpty_USART` flag in the status register. When reading this register, the content of the RX buffer is returned. Additionally, a read access to this register clears the `RxFull_USART` flag in the status register. **Note:** When writing to this register when TX buffer is full, the TX buffer keeps its contents and the written byte is rejected. This is signaled by the `WrCollision_USART` flag in the status register. **Note**: Write access is only possible when the module is enabled.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

4.11.4.4. Register "Z2_USARTIRQEN" – Interrupt Enable

Table 4.60 Register Z2_USARTIRQEN – system address 4000 1C4C$_{HEX}$

Name	Bits	Default	Access	Description
RxOverflow_USART_irq	[0]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
RxFull_USART_irq	[1]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
TxEmpty_USART_irq	[2]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
WrCollision_USART_irq	[3]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
StartErr_irq	[4]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
TxSrEmpty_irq	[5]	0_{BIN}	RW	When set to 1, the corresponding status bit is allowed to drive the IRQ output.
Unused	[31:6]	$0…0_{BIN}$	RO	Unused; always write as 0.

4.11.4.5. Register "Z2_USARTCLK1" and "Z2_USARTCLK2 – Baud Rate Configuration

Table 4.61 Register Z2_USARTCLK1 – system address 4000 1C50$_{HEX}$

Name	Bits	Default	Access	Description
crLsb	[7:0]	FF_{HEX}	RW	Configuration of baud rate. baud rate = clk / ({crMsb, crLsb}+1) **Note:** Change only when module is disabled. **Note**: Do not program values less than 4 for synchronous modes. **Note:** Do not program values less than 3 for asynchronous modes.
Unused	[31:8]	$00\ 0000_{HEX}$	RO	Unused; always write as 0.

Table 4.62 Register Z2_USARTCLK2 – system address 4000 1C54$_{HEX}$

Name	Bits	Default	Access	Description
crMsb	[6:0]	$111\ 1111_{BIN}$	RW	Configuration of baud rate. baud rate = clk / ({crMsb, crLsb}+1) **Note:** Change only when module is disabled. **Note**: Do not program values less than 4 for synchronous modes. **Note:** Do not program values less than 3 for asynchronous modes.
Unused	[31:7]	$0…0_{BIN}$	RO	Unused; always write as 0.

5 ESD / EMC

The ZSSC1956 is designed to maximize EM immunity and minimize emissions.

Functional status A: According to specifications; no LIN communication errors; memory content must not be lost; no wake-up from Sleep Mode; no reset.

Functional status B: According to specifications; offset error extended to ≤ 100mA; no LIN communication errors; memory content must not be lost; no wake-up from Sleep Mode; no reset.

Functional status C: Measurement tolerance beyond specifications; LIN communication errors allowed; memory content must not be lost; reset allowed.

During EM exposure, all functions perform as designed; after exposure, all functions return automatically to within normal limits; memory functions always remain in functional status A.

5.1. Electrostatic Discharge

ESD protection according to AEC-Q100 Rev. G

No.	Parameter	Condition	Min	Max	Unit
1.	ESD, LIN on system level [1)]	IEC 61000-4-2	±6		kV
2.	ESD, BAT+ on system level	IEC 61000-4-2	±6		kV
3.	ESD, HBM, all pins	AEC Q 100-002	±2		kV
4.	ESD, CDM, corner pins	AEC Q 100-011	±750		V
5.	ESD, CDM, all other pins	AEC Q 100-011	±500		V
1) With external protection Diode GSOT36					

5.2. Power System Ripple Factor

Component functionality meets these specifications.

U_N = 13.5V
Voltage variation: sine-wave
Amplitude ΔV = ±2V
Frequency range: 50Hz ≤ F ≤ 25kHz *(linear sweep width for 10 min.)*
Ri of output stage ≤ 100mΩ

5.3. Conducted Susceptibility

DPI in accordance with IEC 62132-4:2006; CW and 80% amplitude modulated carrier frequency; modulation frequency 1 kHz, peak conservation. In the range from 1 to 10 MHz, the step is 0.1 MHz; in the range from 10 to 1000 MHz, it is 1MHz.

The dwell time is longer than the response time of the component and is not less than 1s. The test is carried out with power level 1 and power level 2. Functional status A required for both levels. For bus pin “LIN,” the actual OEM hardware requirements are valid.

© 2016 Integrated Device Technology, Inc.

Table 5.1 Conducted Susceptibility

Group	Pin Description	Frequency Range	Power Level 1	Power Level 2
1	Pin connected to vehicle wiring harness.	1 to <10MHz	0.2W	3.7W
		1 to 1000MHz	1.3W	3.7W
2	Pin not connect to vehicle wiring harness: all pins not in group 3 and 4.	1 to 1000MHz	0.05W	0.1W
3	Pin not connected to vehicle wiring harness: one external high impedance pin (e.g., reference).	1 to 1000MHz	0.01W	0.02W

5.4. Conducted Susceptibility on Power Supply Lines

Test in accordance with ISO 7637-2:2004; pulse amplitudes are under no load condition.

Table 5.2 Conducted Susceptibility on Power Supply Lines

Pulse Mode	Voltage	Internal Resistance Generator	Condition	Burst Cycle / Pulse Repetition Time	Functional Status Class
1	-100V	10Ω	5000 pulses	5s	C
2a	+100V	2Ω	5000 pulses	0.2s	A
3a	-150V	50Ω	1h	100ms	A
3b	+100V	50Ω	1h	100ms	A
4	-7V	0.02Ω	5 pulses	1 min	A
5b	Us = 86.5V	1Ω	5 pulses	1 min	C
	Us* = 26.5V		td=400ms		

5.5. Conducted Susceptibility on Signal Lines

The tests are in accordance with ISO 7637-3:2004. The direct capacitor coupling (DCC) method is used with the functional status A. The pulse amplitudes are under no load condition.

Table 5.3 Conducted Susceptibility on Signal Lines

Pulse Mode	Voltage	Internal Resistance Generator	Coupling Capacitor	Condition	Burst Cycle / Pulse Repetition Time
Fast pulse a	-200V	50Ω	100pF	10 min	100ms
Fast pulse b	+200V	50Ω	100pF	10 min	100ms

© 2016 Integrated Device Technology, Inc.

5.6. Conducted Emission

The tests are in accordance with IEC 61967-4:2002. In the whole frequency range of 0.1 to 1000MHz, a peak detector with a bandwidth of 9kHz (measuring receiver with step 5 kHz) or 10 kHz (spectrum analyzer) is used. The measuring or sweep time is selected in such a way that a longer time will not result in a change of more than 1dB in the measured emission. For the 150Ω method for each pin, the pin-group must be defined. With the 1Ω method, the RF current is measured.

Table 5.4 Conducted Emission

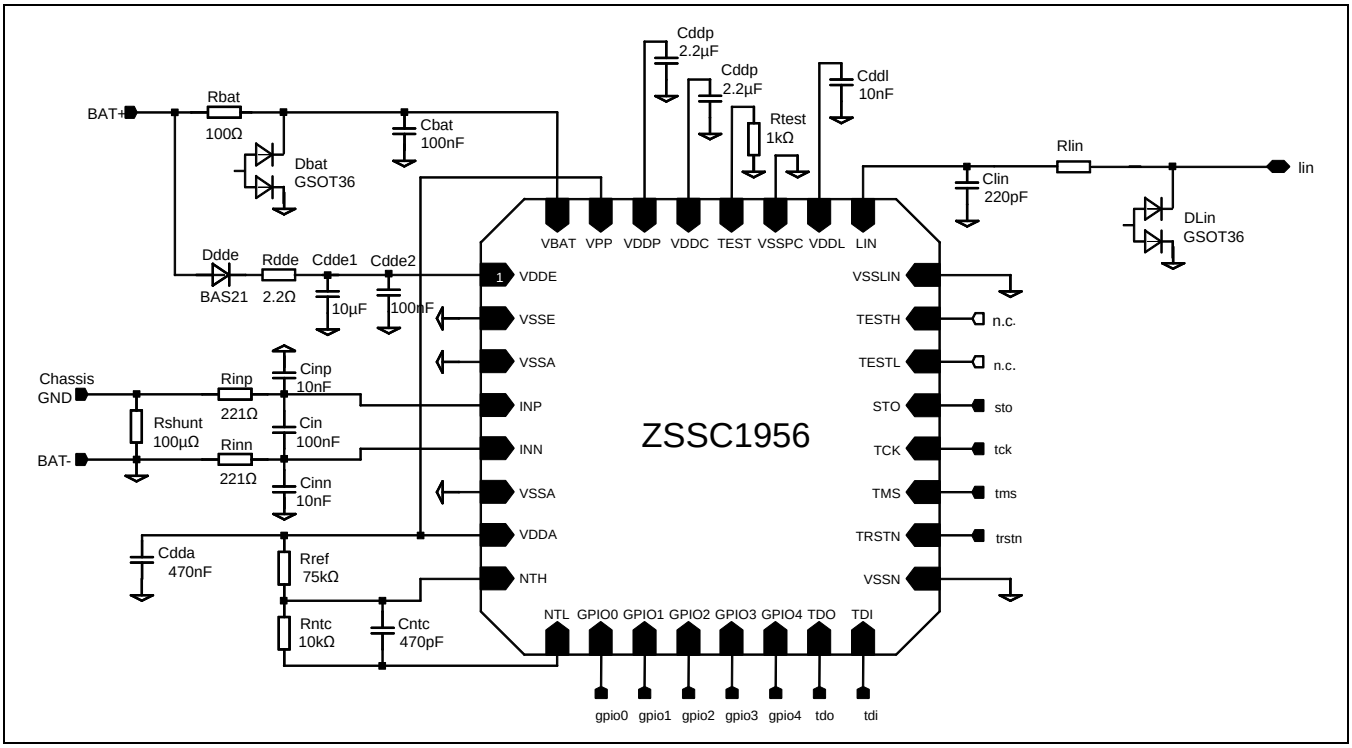
Group	Pin Description	Limit	Method
		15 K 11 m O	1Ω
	Pin connected to vehicle wiring harness:		
A1	Power supply	H 10 k N	150Ω
A2	Analog, static	H 10 k N	150Ω
A3	Digital, PWM	13 H 10 k N	150Ω
	Pin not connected to vehicle wiring harness:		
B1	Power supply	H 10 k M	150Ω
B2	Analog, static, test pin, not connected	H I M	150Ω
B2short	Pin group B connected to trace shorter than 1cm	H g M	150Ω
B3	Digital, PWM	6 e M	150Ω
B4	Oscillator	E I M	150Ω

5.7. Application Circuit Example for EMC Conformance

The final application may require adaption of the external circuit for EMC compliance in the target system as shown in Figure 5.1.

Depending on the application, the resistor Rlin is either a BLM21AG221SN1D ferrite bead or a 20Ω resistor as shown in this application circuit example.

Figure 5.1 Example Application Circuit



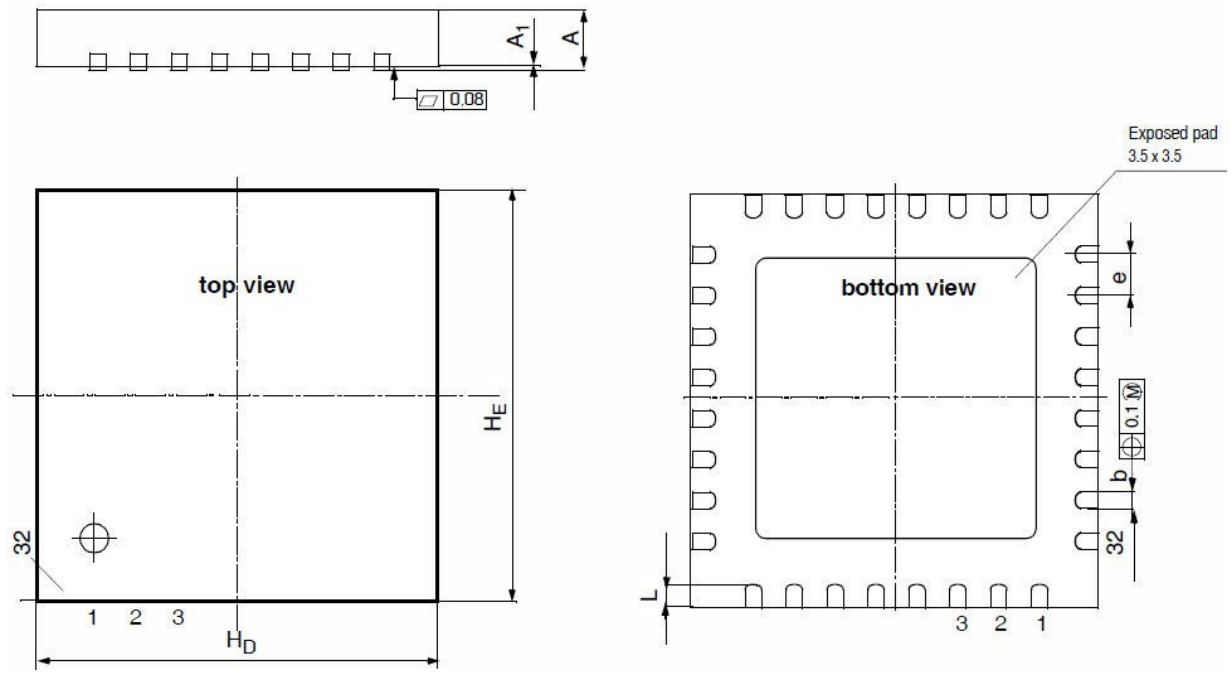
6 Pin Configuration and Package

Table 6.1 IC Pins

PIN	Signal	Description
1	VDDE	Power supply
2	VSSE	Power ground
3	VSSA	Analog voltage ground
4	INP	Positive input for current channel
5	INN	Negative input for current channel
6	VSSA	Analog voltage ground
7	VDDA	Analog voltage supply
8	NTH	Positive input for the temperature channel
9	NTL	Negative input for the temperature channel
10	GPIO0	General purpose I/O
11	GPIO1	General purpose I/O
12	GPIO2	General purpose I/O
13	GPIO3	General purpose I/O
14	GPIO4	General purpose I/O
15	TDO	JTAG output
16	TDI	JTAG input
17	VSSN	Digital voltage ground
18	TRSTN	JTAG input
19	TMS	JTAG input
20	TCK	JTAG input
21	STO	Test data output
22	TESTL	No connection
23	TESTH	No connection
24	VSSLIN	Ground for LIN
25	LIN	LIN input
26	VDDL	SBC core supply and MCU RAM supply
27	VSSPC	Ground for MCU (periphery and core blocks)
28	TEST	Test input
29	VDDC	MCU core supply voltage
30	VDDP	Supply voltage I/O
31	VPP	OTP programming voltage
32	VBAT	Input for battery voltage monitor

© 2016 Integrated Device Technology, Inc.

Figure 6.1 PQFN32 Package Drawing of the ZSSC1956



Dimension	Minimum (mm)	Maximum (mm)
A	0.80	0.90
A_1	0.00	0.05
b	0.20	0.30
e	0.5 nominal	
H_D	4.90	5.10
H_E	4.90	5.10
L	0.30	0.50

7 Ordering Information

Product Sales Code	Description	Package
ZSSC1956BA3R	ZSSC1956 battery sensing IC – temperature range -40°C to +125°C	PQFN32 5x5 mm (reel)
ZSSC1956KIT V1.0	ZSSC1956 Evaluation Kit: modular evaluation and development board for ZSSC1956, IC samples, and USB cable, (software and documentation can be downloaded from the product page at www.IDT.com/ZSSC1956)	

8 Related Documents

8.1. IDT Documents

Document
ZSSC1956 Feature Sheet
*IDT ARM® Cortex™ M0 User Guide **
*ZSSC1956 Evaluation Kit Operation Manual **

Visit the ZSSC1956 product page (www.IDT.com/ZSSC1956) or contact your nearest sales office for the latest version of these documents.

* Note: Documents marked with an asterisk (*) require a login account for access on the web.

8.2. Third-Party Related Documents

Document
LIN Specification Package Revision 2.2 (copyright LIN Consortium)

9 Glossary

Term	Description
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
BIST	Built-in Self-Test
DAP	Debug Access Port
DHCSR	Debug Halting Control and Status Register
EBS	Earliest Bit Sample
ECC	Error Correction Code
FIFO	First In, First Out
FSR	Full Scale Range
IBS	Intelligent Battery Sensor IC
JTAG	Joint Test Action Group
LIN	Local Interconnect Network
LBS	Latest Bit Sample
LSB	Least Significant Bit or Byte Depending on Context
MCU	Microcontroller Unit
MPX	Multiplexer
MRCS	Multiple Results per Conversion Sequence
MSB	Most Significant Bit or Byte Depending on Context
NMI	Non-maskable Interrupt
NTC	Negative Temperature Coefficient
NVIC	Nested Vectored Interrupt Controller
OTP	One Time Programmable Memory
PA-C	Preamplifier for Current
PA-T	Preamplifer for Temperature
PGA	Programmable Gain Amplifier
PMU	Power Management Unit
PID	Protected Identifier
POR	Power-On-Reset
PPB	Private Peripheral Bus

© 2016 Integrated Device Technology, Inc.

Term	Description
PTAT	Proportional to Absolute Temperature
RC	Read Clear
RO	Read Only
RW	Readable and Writable
SBC	System Basis Chip
SCI	Serial Communication Interface
SDM	Sigma Delta Modulator
SMU	System Management Unit
SPI	System Packet Interface
SRCS	Single Result per Conversion Sequence
UART	Universal Asynchronous Receiver/Transmitter
W1C	Write 1 to Clear
WO	Write Only

10 Document Revision History

Revision	Date	Description
1.04	May 18, 2014	First release.
2.00	August 29, 2014	Ordering info updated from xxAA2R to xxBA3R. Updated SPI1 Master (SPI1 to SPIB8); section 4.8. Updated LIN (SW-LIN to AHB-LIN), section 4.7. Parameter 1.2.3 and 1.2.4 updated/changed.
2.01	December 4, 2014	Renamed SW-LIN CTRL as ahbLIN; removed dotted box around ahbBLIN and Master SPIB8 in Figure 2.4. Changed section 4.7 title from “Software Controlled LIN Controller (ahbLIN)” to “LIN Communication Control Logic (ahbLIN).” Section SPIB8 was moved forward by one section (now 4.8). All instances of SW-LIN were replaced by ahbLIN. Update for contact information.
	January 29, 2016	Changed to IDT branding.

Corporate Headquarters
6024 Silver Creek Valley Road
San Jose, CA 95138
www.IDT.com

Sales
1-800-345-7015 or 408-284-8200
Fax: 408-284-2775
www.IDT.com/go/sales

Tech Support
www.IDT.com/go/support

DISCLAIMER Integrated Device Technology, Inc. (IDT) reserves the right to modify the products and/or specifications described herein at any time, without notice, at IDT's sole discretion. Performance specifications and operating parameters of the described products are determined in an independent state and are not guaranteed to perform the same way when installed in customer products. The information contained herein is provided without representation or warranty of any kind, whether express or implied, including, but not limited to, the suitability of IDT's products for any particular purpose, an implied warranty of merchantability, or non-infringement of the intellectual property rights of others. This document is presented only as a guide and does not convey any license under intellectual property rights of IDT or any third parties.

IDT's products are not intended for use in applications involving extreme environmental conditions or in life support systems or similar devices where the failure or malfunction of an IDT product can be reasonably expected to significantly affect the health or safety of users. Anyone using an IDT product in such a manner does so at their own risk, absent an express, written agreement by IDT.

Integrated Device Technology, IDT and the IDT logo are trademarks or registered trademarks of IDT and its subsidiaries in the United States and other countries. Other trademarks used herein are the property of IDT or their respective third party owners. For datasheet type definitions and a glossary of common terms, visit www.idt.com/go/glossary. All contents of this document are copyright of Integrated Device Technology, Inc. All rights reserved.